Title
On Building Generalizable Learning Agents

Author
Wu, Yi

Publication Date
2019

Peer reviewed|Thesis/dissertation

On Building Generalizable Learning Agents

by

Yi Wu

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 Stuart Russell, Chair
Professor Pieter Abbeel
Associate Professor Javad Lavaei
Assistant Professor Aviv Tamar

Fall 2019

On Building Generalizable Learning Agents

Abstract

On Building Generalizable Learning Agents

by

Yi Wu

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Stuart Russell, Chair

It has been a long-standing goal in Artificial Intelligence (AI) to build machines that can solve tasks that humans can. Thanks to the recent rapid progress in data-driven methods, which train agents to solve tasks by learning from massive training data, there have been many successes in applying such learning approaches to handle and even solve a number of extremely challenging tasks, including image classification, language generation, robotics control, and several complex multi-player games. The key factor for all these data-driven successes is that the trained agents can generalize to test scenarios that are unseen during training. This generalization capability is the foundation for building any practical AI system.

This thesis studies generalization, the fundamental challenge in AI, and proposes solutions to improve the generalization performances of learning agents in a variety of domains. We start by providing a formal formulation of the generalization problem in the context of reinforcement learning and proposing 4 principles within this formulation to guide the design of training techniques for improved generalization. We validate the effectiveness of our proposed principles by considering 4 different domains, from simple to complex, and developing domain-specific techniques following these principles. Particularly, we begin with the simplest domain, i.e., path-finding on graphs (Part I), and then consider visual navigation in a 3D world (Part II) and competition in complex multi-agent games (Part III), and lastly tackle some natural language processing tasks (Part IV). Empirical evidences demonstrate that the proposed principles can generally lead to much improved generalization performances in a wide range of problems.

To my beloved, Shihui Li.

Contents

List of Figures

List of Tables

Acknowledgments

I feel tremendously grateful to my advisor, Stuart Russell, for his support and guidance on both research and life throughout my PhD career, for letting me truly understand what I should pursue as a PhD student, and for making me become a better researcher. I also want to express my sincere gratitude and appreciation to my committee members, Pieter Abbeel and Aviv Tamar, who brought me into the area of deep reinforcement learning and keep acting just as my co-advisors. Their advice and support make this thesis possible. I would also thank Javad Lavaei for serving on my dissertation committee as well as Sergey Levine and Trevor Darrell for serving on my qualifying exam committee.

I would sincerely thank some of my mentors and friends, Lei Li, Ras Bodik, Fei Fang, and Danqi Chen, who offer me generous support during my PhD career. I feel fortunate to meet Lei Li for his mentorship, for our long-lasting friendship, and especially for providing me with the opportunity to start my first research project with Stuart. I would thank Fei Fang not only for being my fantastic collaborators and close friend but also for her selfless advice on my career and life. Ras taught me how to present a research project with great patient and raised my confidence in the early stage of my PhD career. His support made my first published work at Berkeley possible. Danqi continuously provided valuable advice to me from my college study to my faculty application. She is also the person who brought me to NLP research.

I deeply appreciate the mentorship from my fabulous internship/project mentors, Igor Mordatch, David Bamman, Yuandong Tian and Georgia Gkioxari. I feel fortunate to have the chance to work with my awesome collaborators, Ryan Lowe, Yuxin Wu, Dave Moore, as well as Tongzhou Wang, the best undergraduate student I have ever mentored. I also appreciate the efforts of my other co-authors, Garrett Thomas, Sergey Levine, Jean Harb, Xinyue Cui and Honghua Dong. In addition, I would like to express my appreciation to all my lovely friends for cheering me up in my daily life.

Finally, I would like to thank my parents, Ru Zhang and Pengfei Wu, for their support, and, most importantly, Shihui Li, for being my co-author, my source of happiness, and my wife.

Chapter 1

Introduction

1.1 Generalization in Artificial Intelligence

Since the term "Artificial Intelligence" was invented at the Dartmouth conference in 1956, tremendous efforts have been constantly devoted to the mission of building artificial agents capable of solving tasks that human beings can. Despite that the original proposal stated a 2-month estimate for a success [McCarthy et al., 2006], it turns out that achieving Artificial General Intelligence, a.k.a. AGI, still remains an open challenge for the whole research community till the current moment. Nevertheless, in the recent decade, there are convincing evidences that we are indeed approaching our ultimate goal of AGI, largely thanks to the remarkable applications of data-driven methods, where agents directly learn to solve tasks from massive data. Particularly, with the advances in deep learning techniques [LeCun et al., 2015], we have successfully trained agents to tackle (and even solve) a lot of real-world challenging tasks from a variety of domains: for example, we can build practical visual systems for image classification [Krizhevsky et al., 2012, He et al., 2016b], segmentation [Long et al., 2015, He et al., 2017] and editing [Goodfellow et al., 2014a, Zhu et al., 2017a, Brock et al., 2019]; we can train AI to generate extremely fluent natural languages [Devlin et al., 2019, Radford et al., 2019] and build high-quality machine translation systems [Wu et al., 2016b]; we can even combine deep learning and reinforcement learning (a.k.a. deep reinforcement learning) to achieve super-human performances in video games [Mnih et al., 2015, Schulman et al., 2015, OpenAI et al., 2019b], Go [Silver et al., 2016], Poker [Brown and Sandholm, 2019] and build real-world robotic systems for navigation [Gupta et al., 2017b, Mirowski et al., 2018], object manipulation [Levine et al., 2016] or solving a Rubik's cube [OpenAI et al., 2019a]. Hence, today's frontier of AI with all these achievements has greatly consolidated our optimism about future success in AGI.

The key ingredient leading to the successes of those recent deep learning applications is that the agents that learn strategies from training data can generalize to unseen scenarios surprisingly well: image classifier trained on ImageNet dataset [Deng et al., 2009] can correctly classify unseen test images [He et al., 2016b]; Google's machine translation product can

handle massive unseen online queries [Wu et al., 2016b]; AIs only trained by self-competition can beat human world champions in Go [Silver et al., 2018] and Dota2 [OpenAI et al., 2019b]. Such generalization ability to bridge the gap between training and testing is a fundamental requirement for building any practical AI system. Although we do have some positive examples as mentioned, generalization is still one of the biggest challenges in AI with countless failure examples for even cutting-edge AI systems: neural networks can produce completely wrong outputs when just a single pixel is modified in the input image [Goodfellow et al., 2014b] or a single word is replaced in the input natural language sentence [Zhang et al., 2019]; in multi-agent physical games, the performance of trained agents can significantly degenerate when competing against an opponent with behavior completely different from its training partners [Al-Shedivat et al., 2018, Gleave et al., 2019]; many robotics systems that are trained in simulators can be brittle when deployed to the real world [Tobin et al., 2017, Peng et al., 2018a, OpenAI et al., 2019a]; DeepMind's StarCraft II AI system, AlphaStar, is still much weaker than the best human professional players [Vinyals et al., 2019]; although Uber's autonomous driving system has passed rigorous tests, it still caused a fatality event in March, 2018[1]; in Chapter 2, we will show that even in an extremely simple grid world task, applying deep learning methods in a straightforward way does not necessarily imply generalization.

Generalization is a fundamental yet extremely challenging problem in AI. This thesis focuses on this challenge in several concrete domains, including path-finding on graphs, embodied agents in 3D world, complex multi-agent games, and natural language processing. We will provide systematic solutions to improve the generalization abilities of learned agents for solving a variety of tasks in these domains. I hope this thesis can serve as a solid step towards practical AGI in the future.

1.2 Problem Formulation

In this section, we will describe the focused generalization problem more precisely. At a high level, let's assume there is a distribution of task scenarios and we want the learning agents to successfully solve the task in all different scenarios. For example, in video games, the scenarios can be different game states and opponent strategies; in robotics manipulation, the scenarios can be different object configurations; in self-driving systems, the scenarios can be different urban and road conditions. Suppose the scenario distribution is denoted by $\mathcal{C}$, the learning agents parameterized by parameter θ is denoted by $\mu(\theta)$ and there is a performance measure on how well the agent solves the task in a particular scenario $c \in \mathcal{C}$ denoted by $M(\mu(\theta); c)$. Then the mathematical objective for building a generalizable learning agent is to find an optimal parameter $\theta^\star$ such that

$$\theta^\star = \arg\max_\theta \mathbb{E}_{c\in\mathcal{C}} \left[M(\mu(\theta); c) \right]. \tag{1.1}$$

In practice, the standard data-driven paradigm is to generate two separate scenario sets, i.e., a training set $\mathcal{C}_{\text{train}}$ and a testing set $\mathcal{C}_{\text{test}}$, then search for the optimal parameter $\theta^\star$ only

[1] https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities

on $\mathcal{C}_{\text{train}}$, and aim for the best performance on $\mathcal{C}_{\text{test}}$. In this case, our true objective becomes

$$\max \mathbb{E}_{c\in\mathcal{C}_{\text{test}}}\left[M(\mu(\hat{\theta}^\star);c)\right], \quad \text{where} \quad \hat{\theta}^\star = \arg\max_\theta \mathbb{E}_{c\in\mathcal{C}_{\text{train}}}\left[M'(\mu(\theta);c)\right], \tag{1.2}$$

where $M'(\cdot)$ denotes the performance measure used at training time: it can be the same as the true measure $M(\cdot)$ or slightly different for the purpose of better generalization.

There are a number of ways to optimize Equation 1.2: we can improve the agent representation $\mu(\theta)$; we can adjust the training measure function $M'(\cdot)$ or use different approaches to search for the best parameter $\hat{\theta}^\star$; we can also augment the training set $\mathcal{C}_{\text{train}}$. The techniques proposed in this thesis all fall into these categories. More methodology details will be presented in Section 1.3.

Key Assumptions

Equation 1.2 provides a very general formulation of generalization. To be more specific, in this thesis, we primarily focus on the setting of reinforcement learning: a scenario c corresponds to a particular Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, P(s'|s,a), r(s,a))$ where $s \in \mathcal{S}$ denotes state space, $a \in \mathcal{A}$ denotes the action space, $P(s'|s,a)$ denotes the transition probability from state s to s' after action a is taken, and $r(s,a)$ denotes the reward to the agent in state s after action a. $\mu(\theta)$ here is called the policy, which takes in the agent's observation $o(s)$ in state s and produces an action $\mu(o(s);\theta) \in \mathcal{A}$. The performance measure becomes the accumulative reward within the horizon H, namely $\mathbb{E}_{s_t}\left[\sum_{t=0}^{H} r(s_t,a_t)|_{a_t=\mu(o(s_t);\theta)}\right]$. For more rigorous definitions, please refer to [Sutton and Barto, 1998]. In each following chapter, these definitions will be recalled and slightly adjusted according to different domain specifications. Note that in the reinforcement learning setting, supervised learning tasks (e.g., image classification) can be viewed as a special case where the agent only needs to make a single action and the reward is simply an indicator function of correctness.

In this thesis, there are two important assumptions:

- MDPs in $\mathcal{C}$ share the same observation space $\mathcal{S}$ and action space $\mathcal{A}$. This simply implies that the input and output space of the policy are consistent across all the scenarios. For example, when building a game AI, no matter which scenario the agent is in, the input is always a game state; for a robotics system, the input can be the sensor signals and the action can be the control, which is always unchanged for a particular robot. Even if some actions become infeasible in some states, we can simply set the associated reward to be negative infinity.

- $\mathcal{C}_{\text{train}}$ and $\mathcal{C}_{\text{test}}$ are both *uniformly* drawn from $\mathcal{C}$. This critical assumption follows the standard assumption in statistical machine learning [James et al., 2013]. It implies that the training scenarios and testing scenarios are sharing some essential common structures. Equivalently, it states that the testing scenarios are never too different from training ones. For example, for autonomous driving systems, a valid setting is that we

train the self-driving car in a lot of cities in the U.S. and particularly test the trained agent in San Francisco. In contrast, if we train the self-driving agent on Earth, testing it on Mars will be beyond our focus in this thesis since the physics on Mars is too different from Earth to follow the same distribution assumption.

Position in Existing Literature

Generalization is the fundamental problem in machine learning, which has been comprehensively studied for decades [James et al., 2013]. For reinforcement learning, although the majority of works in the domain primarily focus on the classical formulation, i.e., learning the optimal policy in a particular MDP, which is essentially training rather than generalization, how to learn policies that can produce good actions on states not covered in training experiences has been also considered since 1980s [Kaelbling et al., 1996]. In the very early stage of studying generalization in reinforcement learning, efforts are put on generalizing over the large state space within an MDP, which is now typically handled by using a function approximator to represent a policy. Herein, we consider generalization over a distribution of scenarios (MDPs). Such a notion was originally proposed in [Caruana, 1997], called *multitask learning*. In this thesis, the problem we tackle is conceptually very similar to the multitask setting but I personally prefer to use a much more general terminology *scenario* instead of *task* since it is often non-trivial to distinguish the exact boundary of reinforcement learning tasks in high-dimensional complex environments. For example, in video games, we want a learned agent to generalize to different game stages/configurations but the "task" itself is still to win the game; autonomous driving systems always aim to transport passengers safely while its generalization focus is on different surrounding situations.

At a high level, the generalization problem falls into the scope of *transfer learning* [Pratt, 1993, Pan and Yang, 2009], which aims to utilize a learned model for one problem to solve another different but related problem. It is sometimes called domain adaptation when the goal is specifically to transfer a learned model trained on one data distribution to another testing data distribution [Patel et al., 2015]. Transfer learning has also been widely studied in the reinforcement learning setting [Taylor and Stone, 2009]. As a very general paradigm, transfer learning does not assume the training and test scenarios are drawn from the same distribution (our second assumption). Instead, it primarily considers the problem of how to bridge the gap between distribution shifts between training and testing data.

Another related field is *learning to learn* [Schmidhuber, 1995, Thrun and Pratt, 2012], which aims to build an agent that learns how to learn and adapt in new situations. It is also called *meta-learning* [Schaul and Schmidhuber, 2010, Vilalta and Drissi, 2002, Finn, 2018]. The goal of learning to learn is essentially the same as our focus, namely building agents that can perform well on unseen testing scenarios. The difference is that in the learning to learn setting, when the learned agent is deployed at test time, the agent will be typically given a short period of "free time" to further learn or adapt in the testing environment. While in the aforementioned generalization formulation, we want the learned agent to *immediately* work in

test cases without an explicit adaptation procedure. From this perspective, the formulation here is also related to *zero-shot learning* [Larochelle et al., 2008, Palatucci et al., 2009].

Additionally, in multi-agent games, an agent should not only behave well on different game configurations but also respond optimally to other agents' strategies in the game. Learning optimal strategies in multi-agent games has a long history within the scope of algorithmic game theory [Nisan et al., 2007]. In Part III of this thesis, we will leverage several fundamental ideas from algorithmic game theory and design computationally efficient approximate algorithms to build agents that can generalize across different opponent strategies in complex multi-agent games.

1.3 Contributions

This thesis studies the generalization challenge following the aforementioned formulation in Equation 1.2. I propose that improved generalization can be achieved via applying specially designed training techniques under the following 4 design principles:

(1) Incorporate inductive bias into policy representation $\mu(\theta)$ to better capture the underlying problem structure in $\mathcal{C}$;

(2) Augment the training scenarios $\mathcal{C}_{\text{train}}$ to better represent the underlying distribution $\mathcal{C}$;

(3) Utilize a better designed training performance measure $M'(\cdot)$ that encourages the agent to learn to generalize;

(4) Develop better algorithms or training paradigms such that in practice, a better policy parameter $\hat{\theta}^\star$ can be obtained for the problem to solve.

We empirically validate our proposed principles by considering a wide range of specific yet important domains, from simple to hard, and applying solutions in these domains following the above design principles. Experimental results show that these principles can lead to significantly better generalization performances in a wide range of problems. Note that in each specific domain, particular techniques are developed under the guidance of these principles. It is yet to be explored if there exists a unified solution to improve generalization in all kinds of problems, but I believe this thesis is a starting point towards this ultimate goal for eventually solving the generalization challenge.

The following content of this thesis is organized as follows:

- In Part I starts from the simplest domain, i.e., the grid world and graphs. Chapter 2 explores *principle (1)* in a standard navigation task where the agent needs to find a path towards some target location. A standard deep learning paradigm is utilized for training. It can be demonstrated that policies represented by standard feed-forward networks do not generalize well in solving tasks in unseen scenarios. We argue that this is because the policy representation lacks the capability of performing long-term planning when

producing decisions. Therefore, Chapter 2 proposes a new policy representation, called value iteration network, which enables the agents to learn a planning computation for long-term consequences, and shows that the proposed representation leads to significantly better generalization performances. Chapter 2 was originally published as a part of [Tamar et al., 2016], which won the best paper award at NeurIPS 2016.

- Part II considers a more realistic setting, i.e., visual navigation, where an agent takes in visual signals and needs to navigate in a 3D interactive world condition on some instruction. Chapter 3 explores *principle (2)* by proposing a new environment, House3D, which contains thousands of human-designed indoor scenes. House3D is particularly designed for building generalizable visual embodied agents. We further proposed a benchmark task called RoomNav and systematically evaluated a variety of data augmentation and policy representation techniques for the RoomNav task in House3D. Chapter 4 further explores *principle (1)* by proposing a new memory architecture for policy representation, called Bayesian Relational Memory (BRM). BRM allows the agents to effectively plan for the optimal path towards the target based on its past experiences and therefore significantly improves the generalization performances in the RoomNav task. Chapter 3 was originally published at ICLR 2018 workshop track as [Wu et al., 2018]. Chapter 4 was published at ICCV 2019 as [Wu et al., 2019].

- Part III considers mixed cooperative-competitive multi-agent games where an agent should generalize to not only game configurations but also other agents' strategies. Learning good policies in the multi-agent setting is fundamentally more challenging than the single-agent case. Chapter 5 explores *principle (4)* by proposing a new actor-critic multi-agent reinforcement learning algorithm, MADDPG. It also provides preliminary examinations of the generalization performances of learned policies by testing with unseen competitors and suggests an ensemble learning technique to improve generalization. Chapter 6 further explores *principle (3), (4)* by incorporating the *minimax* concept from game theory into the MADDPG algorithm and developing a computationally efficient minimax learning algorithm, M3DDPG, which can significantly improve the generalization performances of learned policies in a variety of complex multi-agent games. Chapter 5 was originally published at NeurIPS 2017 as [Lowe et al., 2017]. Chapter 6 was published at AAAI 2019 as [Li et al., 2019].

- Part IV considers the natural language processing domain. Chapter 7 explores *principle (3)* in task of relation extraction and proposes an adversarial training objective that generally improves the testing performances on a number of neural network models. Chapter 8 explores *principle (2), (4)* in the task of named entity recognition (NER). A classical pipeline for this task is to first learn a probabilistic model of text sequences during training and then perform probabilistic inference on the learned model to produce test predictions. Inspired by the core idea of meta-learning, Chapter 8 proposes a meta-learning paradigm for MCMC inference, which can be directly applied to a trained NER model at test time and significantly boosts its test performances with

little additional computational cost. Chapter 7 was published at EMNLP 2017 as [Wu et al., 2017b]. Chapter 8 was published at NeurIPS 2018 as [Wang et al., 2018a].

Part I

Decision Making on Graphs

Chapter 2

Value Iteration Network

In this chapter, we introduce a novel policy representation, the *value iteration network* (VIN): a fully differentiable neural network with a 'planning module' embedded within. VINs can *learn to plan*, and are suitable for predicting outcomes that involve planning-based reasoning. Key to our approach is a novel *differentiable* approximation of the value-iteration algorithm, which can be represented as a convolutional neural network, and trained end-to-end using standard backpropagation. We evaluate VIN based policies on discrete path-planning domains, including a 2D grid world and a natural language based web search task. We show that by learning an explicit planning computation, VIN policies generalize better to new, unseen domains.

2.1 Motivation

Over the last decade, deep convolutional neural networks (CNNs) have revolutionized supervised learning for tasks such as object recognition, action recognition, and semantic segmentation [Ciresan et al., 2012, Krizhevsky et al., 2012, Farabet et al., 2013, Long et al., 2015]. Recently, CNNs have been applied to reinforcement learning (RL) tasks with visual observations such as Atari games [Mnih et al., 2015], robotic manipulation [Levine et al., 2016], and imitation learning (IL) [Giusti et al., 2016]. In these tasks, a neural network (NN) is trained to represent a *policy* – a mapping from an observation of the system's state to an action, with the goal of representing a control strategy that has good *long-term* behavior, typically quantified as the minimization of a sequence of time-dependent costs.

The *sequential* nature of decision making in RL is inherently different than the one-step decisions in supervised learning, and in general requires some form of *planning* [Bertsekas, 2012]. However, most recent deep RL works [Mnih et al., 2015, Schulman et al., 2015, Levine et al., 2016, Giusti et al., 2016] employed NN architectures that are very similar to the standard networks used in supervised learning tasks, which typically consist of CNNs for feature extraction, and fully connected layers that map the features to a probability distribution over actions. Such networks are inherently *reactive*, and in particular, lack explicit *planning*

computation. The success of reactive policies in sequential problems is due to the *learning algorithm*, which essentially trains a reactive policy to select actions that have good long-term consequences in its training domain.

To understand why planning can nevertheless be an important ingredient in a policy, consider the grid-world navigation task depicted in Figure 2.1 (left), in which the agent can observe a map of its domain, and is required to navigate between some obstacles to a target position. One hopes that after training a policy to solve several instances of this problem with different obstacle configurations, the policy would generalize to solve a different, unseen domain, as in Figure 2.1 (right). However, as we show in our experiments, while standard CNN-based networks can be easily trained to solve a set of such maps, they do not generalize well to new tasks outside this set, because they do not understand the goal-directed nature of the behavior. This observation suggests that the computation learned by reactive policies is different from planning, which is required to solve a new task[1].

In this chapter, we propose a NN-based policy that can effectively *learn to plan*. Our model, termed a *value-iteration network* (VIN), has a differentiable 'planning program' embedded within the NN structure.

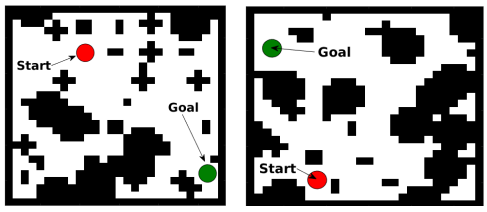


Figure 2.1: Two instances of a grid-world domain. Task is to move to the goal between the obstacles.

The key to our approach is an observation that the classic value-iteration (VI) planning algorithm [Bellman, 1957, Bertsekas, 2012] may be represented by a specific type of CNN. By embedding such a VI network module inside a standard feed-forward classification network, we obtain a NN model that can learn the parameters of a planning computation that yields useful predictions. The VI block is differentiable, and the whole network can be trained using standard backpropagation. This makes our policy simple to train using standard RL and IL algorithms, and straightforward to integrate with NNs for perception and control.

Connections between planning algorithms and recurrent NNs were previously explored by Ilin et al. [Ilin et al., 2007]. Our work builds on related ideas, but results in a more broadly applicable policy representation. Our approach is different from model-based RL [Schmidhuber, 1990, Deisenroth and Rasmussen, 2011], which requires system identification to map the observations to a dynamics model, which is then solved for a policy. In many applications, including robotic manipulation and locomotion, accurate system identification is difficult, and modelling errors can severely degrade the policy performance. In such domains, a model-free approach is often preferred [Levine et al., 2016]. Since a VIN is just a NN policy, it can be trained model free, without requiring explicit system identification. In addition,

[1]In principle, with enough training data that covers all possible task configurations, and a rich enough policy representation, a reactive policy can learn to map each task to its optimal policy. In practice, this is often too expensive, and we offer a more data-efficient approach by exploiting a flexible prior about the planning computation underlying the behavior.

the effects of modelling errors in VINs can be mitigated by training the network end-to-end, similarly to the methods in [Joseph et al., 2013, Guo et al., 2016].

We demonstrate the effectiveness of VINs in various problems, from gridworld path-finding to natural language based decision making in the WebNav challenge [Nogueira and Cho, 2016]. After training, the policy learns to map an observation to a planning computation relevant for the task, and generate action predictions based on the resulting plan. As we demonstrate, this leads to policies that *generalize better* to new, unseen, task instances.

2.2 Background

In this section we provide background on planning, value iteration, CNNs, and policy representations for RL and IL. In the sequel, we shall show that CNNs can implement a particular form of planning computation similar to the value iteration algorithm, which can then be used as a policy for RL or IL.

Value Iteration: A standard model for sequential decision making and planning is the Markov decision process (MDP) [Bellman, 1957, Bertsekas, 2012]. An MDP M consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, a reward function $R(s,a)$, and a transition kernel $P(s'|s,a)$ that encodes the probability of the next state given the current state and action. A policy $\pi(a|s)$ prescribes an action distribution for each state. The goal in an MDP is to find a policy that obtains high rewards in the *long term*. Formally, the *value* $V^\pi(s)$ of a state under policy π is the expected discounted sum of rewards when starting from that state and executing policy π, $V^\pi(s) \doteq \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) | \, s_0 = s\right]$, where $\gamma \in (0,1)$ is a discount factor, and $\mathbb{E}^\pi$ denotes an expectation over trajectories of states and actions $(s_0, a_0, s_1, a_1 \dots)$, in which actions are selected according to π, and states evolve according to the transition kernel $P(s'|s,a)$. The optimal value function $V^*(s) \doteq \max_\pi V^\pi(s)$ is the maximal long-term return possible from a state. A policy π^* is said to be optimal if $V^{\pi^*}(s) = V^*(s) \;\; \forall s$. A popular algorithm for calculating V^* and π^* is value iteration (VI):

$$V_{n+1}(s) = \max_a Q_n(s,a) \quad \forall s, \quad \text{where} \quad Q_n(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_n(s'). \quad (2.1)$$

It is well known that the value function V_n in VI converges as $n \to \infty$ to V^*, from which an optimal policy may be derived as $\pi^*(s) = \arg\max_a Q_\infty(s,a)$.

Convolutional Neural Networks (CNNs) are NNs with a particular architecture that has proved useful for computer vision, among other domains [Fukushima, 1979, LeCun et al., 1998, Ciresan et al., 2012, Krizhevsky et al., 2012]. A CNN is comprised of stacked convolution and max-pooling layers. The input to each convolution layer is a 3-dimensional signal X, typically, an image with l channels, m horizontal pixels, and n vertical pixels, and its output h is a l'-channel convolution of the image with kernels $W^1, \dots, W^{l'}$, $h_{l',i',j'} = \sigma\left(\sum_{l,i,j} W^{l'}_{l,i,j} X_{l,i'-i,j'-j}\right)$, where σ is some scalar activation function. A max-pooling layer selects, for each channel l and pixel i,j in h, the maximum value among its neighbors $N(i,j)$, $h^{maxpool}_{l,i,j} = \max_{i',j' \in N(i,j)} h_{l,i',j'}$. Typically, the neighbors $N(i,j)$ are chosen as a $k \times k$

image patch around pixel i, j. After max-pooling, the image is down-sampled by a constant factor d, commonly 2 or 4, resulting in an output signal with l' channels, m/d horizontal pixels, and n/d vertical pixels. CNNs are typically trained using stochastic gradient descent (SGD), with backpropagation for computing gradients.

Reinforcement Learning and Imitation Learning: In MDPs where the state space is very large or continuous, or when the MDP transitions or rewards are not known in advance, planning algorithms cannot be applied. In these cases, a policy can be *learned* from either expert supervision – IL, or by trial and error – RL. While the learning algorithms in both cases are different, the policy representations – which are the focus of this chapter – are similar. Additionally, most state-of-the-art algorithms such as [Ross et al., 2011a, Mnih et al., 2015, Schulman et al., 2015, Levine et al., 2016] are agnostic to the policy representation, and only require it to be differentiable, for performing gradient descent on some algorithm-specific loss function. Therefore, in this chapter we do not commit to a specific learning algorithm, and only consider the policy.

Let $\phi(s)$ denote an observation for state s. The policy is specified as a parametrized function $\pi_\theta(a|\phi(s))$ mapping observations to a probability over actions, where θ are the policy parameters. For example, the policy could be represented as a neural network, with θ denoting the network weights. The goal is to tune the parameters such that the policy behaves well in the sense that $\pi_\theta(a|\phi(s)) \approx \pi^*(a|\phi(s))$, where π^* is the optimal policy for the MDP, as defined in Section 2.2.

In IL, a dataset of N state observations and corresponding optimal actions $\{\phi(s^i), a^i \sim \pi^*(\phi(s^i))\}_{i=1,\ldots,N}$ is generated by an expert. Learning a policy then becomes an instance of supervised learning [Ross et al., 2011a, Giusti et al., 2016]. In RL, the optimal action is not available, but instead, the agent can act in the world and observe the rewards and state transitions its actions effect. RL algorithms such as in [Sutton and Barto, 1998, Mnih et al., 2015, Schulman et al., 2015, Levine et al., 2016] use these observations to improve the value of the policy.

2.3 The Value Iteration Network Model

In this section we introduce a general policy representation that embeds an explicit *planning module*. As stated earlier, the motivation for such a representation is that a natural solution to many tasks, such as the path planning described above, involves planning on some model of the domain.

Let M denote the MDP of the domain for which we design our policy π. We assume that there is some unknown MDP $\bar{M}$ such that the optimal plan in $\bar{M}$ contains useful information about the optimal policy in the original task M. However, we emphasize that we do not assume to know $\bar{M}$ in advance. Our idea is to equip the policy with the *ability to learn and solve* $\bar{M}$, and to add the solution of $\bar{M}$ as an element in the policy π. We hypothesize that this will lead to a policy that automatically learns a useful $\bar{M}$ to plan on. We denote by $\bar{s} \in \bar{S}$, $\bar{a} \in \bar{A}$, $\bar{R}(\bar{s}, \bar{a})$, and $\bar{P}(\bar{s}'|\bar{s}, \bar{a})$ the states, actions, rewards, and transitions in $\bar{M}$. To facilitate a connection between M and $\bar{M}$, we let $\bar{R}$ and $\bar{P}$ depend on the observation in M,

namely, $\bar{R} = f_R(\phi(s))$ and $\bar{P} = f_P(\phi(s))$, and we will later learn the functions f_R and f_P as a part of the policy learning process.

For example, in the grid-world domain described above, we can let $\bar{M}$ have the same state and action spaces as the true grid-world M. The reward function f_R can map an image of the domain to a high reward at the goal, and negative reward near an obstacle, while f_P can encode deterministic movements in the grid-world that do not depend on the observation. While these rewards and transitions are not necessarily the true rewards and transitions in the task, an optimal plan in $\bar{M}$ will still follow a trajectory that avoids obstacles and reaches the goal, similarly to the optimal plan in M.

Once an MDP $\bar{M}$ has been specified, any standard planning algorithm can be used to obtain the value function $\bar{V}^*$. In the next section, we shall show that using a particular implementation of VI for planning has the advantage of being differentiable, and simple to implement within a NN framework. In this section however, we focus on how to use the planning result $\bar{V}^*$ within the NN policy π. Our approach is based on two important observations. The first is that the vector of values $\bar{V}^*(s) \ \ \forall s$ encodes all the information about the optimal plan in $\bar{M}$. Thus, adding the vector $\bar{V}^*$ as additional features to the policy π is sufficient for extracting information about the optimal plan in $\bar{M}$.

However, an additional property of $\bar{V}^*$ is that the optimal decision $\bar{\pi}^*(\bar{s})$ at a state $\bar{s}$ can depend only on a subset of the values of $\bar{V}^*$, since $\bar{\pi}^*(\bar{s}) = \arg\max_{\bar{a}} \bar{R}(\bar{s}, \bar{a}) + \gamma \sum_{\bar{s}'} \bar{P}(\bar{s}'|\bar{s}, \bar{a}) \bar{V}^*(\bar{s}')$. Therefore, if the MDP has a local connectivity structure, such as in the grid-world example above, the states for which $\bar{P}(\bar{s}'|\bar{s}, \bar{a}) > 0$ is a small subset of $\bar{S}$.

In NN terminology, this is a form of *attention* [Xu et al., 2015], in the sense that for a given label prediction (action), only a subset of the input features (value function) is relevant. Attention is known to improve learning performance by reducing the effective number of network parameters during learning. Therefore, the second element in our network is an attention module that outputs a vector of (attention modulated) values $\psi(s)$. Finally, the vector $\psi(s)$ is added as additional features to a reactive policy $\pi_{\text{re}}(a|\phi(s), \psi(s))$. The full network architecture is depicted in Figure 2.2 (left).

Returning to our grid-world example, at a particular state s, the reactive policy only needs to query the values of the states neighboring s in order to select the correct action. Thus, the attention module in this case could return a $\psi(s)$ vector with a subset of $\bar{V}^*$ for these neighboring states.

Let θ denote all the parameters of the policy, namely, the parameters of f_R, f_P, and π_{re}, and note that $\psi(s)$ is in fact a function of $\phi(s)$. Therefore, the policy can be written in the form $\pi_\theta(a|\phi(s))$, similarly to the standard policy form (cf. Section 2.2). If we could back-propagate through this function, then potentially we could train the policy using standard RL and IL algorithms, just like any other standard policy representation. While it is easy to design functions f_R and f_P that are differentiable (and we provide several examples in our experiments), back-propagating the gradient through the planning algorithm is not trivial. In the following, we propose a novel interpretation of an approximate VI algorithm as a particular form of a CNN. This allows us to conveniently treat the planning module as just another NN, and by back-propagating through it, we can train the whole policy *end-to-end*.

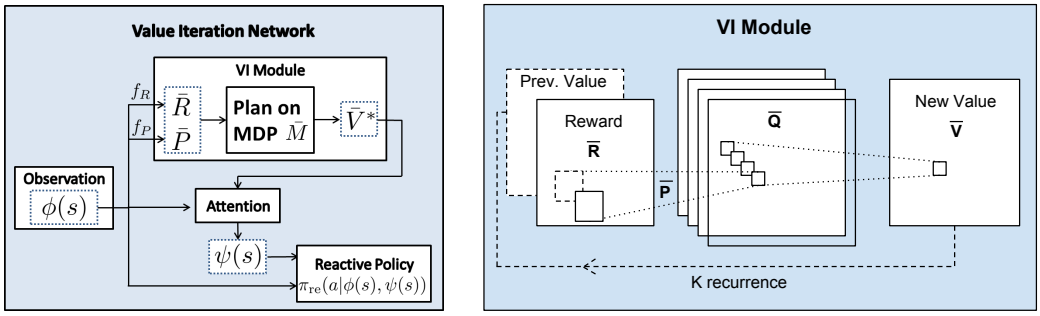


Figure 2.2: Planning-based NN models. Left: a general policy representation that adds value function features from a planner to a reactive policy. Right: VI module – a CNN representation of VI algorithm.

The VI Module

We now introduce the VI module – a NN that encodes a differentiable planning computation.

Our starting point is the VI algorithm (2.1). Our main observation is that each iteration of VI may be seen as passing the previous value function V_n and reward function R through a convolution layer and max-pooling layer. In this analogy, each channel in the convolution layer corresponds to the Q-function for a specific action, and convolution kernel weights correspond to the discounted transition probabilities. Thus by recurrently applying a convolution layer K times, K iterations of VI are effectively performed.

Following this idea, we propose the VI network module, as depicted in Figure 2.2B. The inputs to the VI module is a 'reward image' $\bar{R}$ of dimensions l, m, n, where here, for the purpose of clarity, we follow the CNN formulation and explicitly assume that the state space $\bar{S}$ maps to a 2-dimensional grid. However, our approach can be extended to general discrete state spaces, for example, a graph, as we report in the WikiNav experiment in Section 2.4. The reward is fed into a convolutional layer $\bar{Q}$ with $\bar{A}$ channels and a linear activation function, $\bar{Q}_{\bar{a},i',j'} = \sum_{l,i,j} W^{\bar{a}}_{l,i,j} \bar{R}_{l,i'-i,j'-j}$. Each channel in this layer corresponds to $\bar{Q}(\bar{s},\bar{a})$ for a particular action $\bar{a}$. This layer is then max-pooled along the actions channel to produce the next-iteration value function layer $\bar{V}$, $\bar{V}_{i,j} = \max_{\bar{a}} \bar{Q}(\bar{a},i,j)$. The next-iteration value function layer $\bar{V}$ is then stacked with the reward $\bar{R}$, and *fed back* into the convolutional layer and max-pooling layer K times, to perform K iterations of value iteration.

The VI module is simply a NN architecture that has the capability of performing an approximate VI computation. Nevertheless, representing VI in this form makes *learning* the MDP parameters and reward function natural – by backpropagating through the network, similarly to a standard CNN. VI modules can also be composed hierarchically, by treating the value of one VI module as additional input to another VI module. We further report on this idea in the supplementary material.

Value Iteration Networks

We now have all the ingredients for a differentiable planning-based policy, which we term a value iteration network (VIN). The VIN is based on the general planning-based policy defined above, with the VI module as the planning algorithm. In order to implement a VIN, one has to specify the state and action spaces for the planning module $\bar{S}$ and $\bar{A}$, the reward and transition functions f_R and f_P, and the attention function; we refer to this as the *VIN design*. For some tasks, as we show in our experiments, it is relatively straightforward to select a suitable design, while other tasks may require more thought. However, we emphasize an important point: the reward, transitions, and attention can be defined by parametric functions, and trained with the whole policy[2]. Thus, a rough design can be specified, and then fine-tuned by end-to-end training.

Once a VIN design is chosen, implementing the VIN is straightforward, as it is simply a form of a CNN. The networks in our experiments all required only several lines of Theano [Al-Rfou et al., 2016] code. In the next section, we evaluate VIN policies on various domains, showing that by learning to plan, they achieve a better generalization capability.

2.4 Experiments

In this section we evaluate VINs as policy representations on various domains. Additional experiments investigating RL and hierarchical VINs, as well as technical implementation details are discussed in the supplementary material. Source code is available at `https://github.com/avivt/VIN`.
Our goal in these experiments is to investigate the following questions:

1. Can VINs effectively learn a planning computation using standard RL and IL algorithms?
2. Does the planning computation learned by VINs make them better than reactive policies at generalizing to new domains?

An additional goal is to point out several ideas for designing VINs for various tasks. While this is not an exhaustive list that fits all domains, we hope that it will motivate creative designs in future work.

Grid-World Domain

Our first experiment domain is a synthetic grid-world with randomly placed obstacles, in which the observation includes the position of the agent, and also an image of the map of obstacles and goal position. Figure 2.3 shows two random instances of such a grid-world of size 16×16. We conjecture that by learning the optimal policy for several instances of this

[2]VINs are fundamentally different than inverse RL methods [Neu and Szepesvári, 2007], where transitions are required to be known.

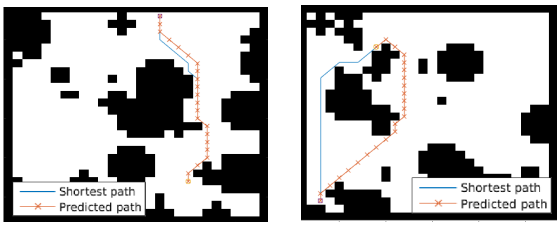


Figure 2.3: Two random instances of the 28×28 synthetic gridworld, with the VIN-predicted trajectories and ground-truth shortest paths between random start and goal positions. In A, the trajectories mostly overlap. Note that the shortest-path trajectory is not unique, and in the left domain the trajectory predicted by the VIN is different than the ground-truth, but of the same length.

domain, a VIN policy would learn the planning computation required to solve a new, unseen, task.

In such a simple domain, an optimal policy can easily be calculated using exact VI. Note, however, that here we are interested in evaluating whether a NN policy, trained using RL or IL, can *learn to plan*. In the following results, policies were trained using IL, by standard supervised learning from demonstrations of the optimal policy. In the supplementary material, we report additional RL experiments that show similar findings.

We design a VIN for this task following the guidelines described above, where the planning MDP $\bar{M}$ is a grid-world, similar to the true MDP. The reward mapping f_R is a CNN mapping the image input to a reward map in the grid-world. Thus, f_R should potentially learn to discriminate between obstacles, non-obstacles and the goal, and assign a suitable reward to each. The transitions $\bar{P}$ were defined as 3×3 convolution kernels in the VI block, exploiting the fact that transitions in the grid-world are local[3]. The recurrence K was chosen in proportion to the grid-world size, to ensure that information can flow from the goal state to any other state. For the attention module, we chose a trivial approach that selects the $\bar{Q}$ values in the VI block for the current state, i.e., $\psi(s) = \bar{Q}(s, \cdot)$. The final reactive policy is a fully connected network that maps $\psi(s)$ to a probability over actions.

We compare VINs to the following NN reactive policies:

CNN network: We devised a CNN-based reactive policy inspired by the recent impressive results of DQN [Mnih et al., 2015], with 5 convolution layers, and a fully connected output. While the network in [Mnih et al., 2015] was trained to predict Q values, our network outputs a probability over actions. These terms are related, since $\pi^*(s) = \arg\max_a Q(s, a)$.
Fully Convolutional Network (FCN): The problem setting for this domain is similar to semantic segmentation [Long et al., 2015], in which each pixel in the image is assigned a semantic label (the action in our case). We therefore devised an FCN inspired by a state-of-the-art semantic segmentation algorithm [Long et al., 2015], with 3 convolution layers, where

[3]Note that the transitions defined this way do not depend on the state $\bar{s}$. Interestingly, we shall see that the network learned to plan successful trajectories nevertheless, by appropriately shaping the reward.

Domain	VIN			CNN			FCN		
	Prediction loss	Success rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.
8×8	0.004	**99.6**%	0.001	0.02	97.9%	0.006	0.01	97.3%	0.004
16×16	0.05	**99.3**%	0.089	0.10	87.6%	0.06	0.07	88.3%	0.05
28×28	0.11	**97**%	0.086	0.13	74.2%	0.078	0.09	76.6%	0.08

Table 2.1: Performance on grid-world domain. Top: comparison with reactive policies. For all domain sizes, VIN networks significantly outperform standard reactive networks. Note that the performance gap increases dramatically with problem size.

the first layer has a filter that spans the whole image, to properly convey information from the goal to every other state.

In Table 2.1 we present the average $0-1$ prediction loss of each model, evaluated on a held-out test-set of maps with random obstacles, goals, and initial states, for different problem sizes. In addition, for each map, a full trajectory from the initial state was predicted, by iteratively rolling-out the next-states predicted by the network. A trajectory was said to succeed if it reached the goal without hitting obstacles. For each trajectory that succeeded, we also measured its difference in length from the optimal trajectory. The average difference and the average success rate are reported in Table 2.1.

Clearly, VIN policies generalize to domains outside the training set. A visualization of the reward mapping f_R (see supplementary material) shows that it is negative at obstacles, positive at the goal, and a small negative constant otherwise. The resulting value function has a gradient pointing towards a direction to the goal around obstacles, thus a useful planning computation was learned. VINs also significantly outperform the reactive networks, and the performance gap increases dramatically with the problem size. Importantly, note that the prediction loss for the reactive policies is comparable to the VINs, although their success rate is significantly worse. This shows that this is not a standard case of overfitting/underfitting of the reactive policies. Rather, VIN policies, by their VI structure, focus prediction errors on less important parts of the trajectory, while reactive policies do not make this distinction, and learn the easily predictable parts of the trajectory yet fail on the complete task.

The VINs have an effective depth of K, which is larger than the depth of the reactive policies. One may wonder, whether any deep enough network would learn to plan. In principle, a CNN or FCN of depth K has the potential to perform the same computation as a VIN. However, it has much more parameters, requiring much more training data. We evaluate this by untying the weights in the K recurrent layers in the VIN. Our results, reported in the supplementary material, show that untying the weights degrades performance, with a stronger effect for smaller sizes of training data.

WebNav Challenge

In the previous experiments, the planning aspect of the task corresponded to 2D navigation. We now consider a more general domain: WebNav [Nogueira and Cho, 2016] – a language based search task on a graph.

In WebNav [Nogueira and Cho, 2016], the agent needs to navigate the links of a website towards a goal web-page, specified by a short 4-sentence query. At each state s (web-page), the agent can observe average word-embedding features of the state $\phi(s)$ and possible next states $\phi(s')$ (linked pages), and the features of the query $\phi(q)$, and based on that has to select which link to follow. In [Nogueira and Cho, 2016], the search was performed on the Wikipedia website. Here, we report experiments on the 'Wikipedia for Schools' website, a simplified Wikipedia designed for children, with over 6000 pages and at most 292 links per page.

In [Nogueira and Cho, 2016], a NN-based policy was proposed, which first learns a NN mapping from $(\phi(s), \phi(q))$ to a hidden state vector h. The action is then selected according to $\pi(s'|\phi(s), \phi(q)) \propto \exp\left(h^\top \phi(s')\right)$. In essence, this policy is reactive, and relies on the word embedding features at each state to contain meaningful information about the path to the goal. Indeed, this property naturally holds for an encyclopedic website that is structured as a tree of categories, sub-categories, sub-sub-categories, etc.

We sought to explore whether planning, based on a VIN, can lead to better performance in this task, with the intuition that a plan on a simplified model of the website can help guide the reactive policy in difficult queries. Therefore, we designed a VIN that plans on a small subset of the graph that contains only the 1st and 2nd level categories ($< 3\%$ of the graph), and their word-embedding features.

Designing this VIN requires a different approach from the grid-world VINs described earlier, where the most challenging aspect is to define a meaningful mapping between nodes in the true graph and nodes in the smaller VIN graph. For the reward mapping f_R, we chose a weighted similarity measure between the query features $\phi(q)$, and the features of nodes in the small graph $\phi(\bar{s})$. Thus, intuitively, nodes that are similar to the query should have high reward. The transitions were fixed based on the graph connectivity of the smaller VIN graph, which is known, though different from the true graph. The attention module was also based on a weighted similarity measure between the features of the possible next states $\phi(s')$ and the features of each node in the simplified graph $\phi(\bar{s})$. The reactive policy part of the VIN was similar to the policy of [Nogueira and Cho, 2016] described above. Note that by training such a VIN end-to-end, we are effectively learning how to exploit the small graph for doing better planning on the true, large graph.

Both the VIN policy and the baseline reactive policy were trained by supervised learning, on random trajectories that start from the root node of the graph. Similarly to [Nogueira and Cho, 2016], a policy is said to succeed a query if all the correct predictions along the path are within its top-4 predictions.

After training, the VIN policy performed mildly better than the baseline on 2000 held-out

test queries when starting from the root node, achieving 1030 successful runs vs. 1025 for the baseline. However, when we tested the policies on a harder task of starting from a random position in the graph, VINs significantly outperformed the baseline, achieving 346 successful runs vs. 304 for the baseline, out of 4000 test queries. These results confirm that indeed, when navigating a tree of categories from the root up, the features at each state contain meaningful information about the path to the goal, making a reactive policy sufficient. However, when starting the navigation from a different state, a reactive policy may fail to understand that it needs to first go back to the root and switch to a different branch in the tree. Our results indicate such a strategy can be better represented by a VIN.

We remark that there is still room for further improvements of the WebNav results, e.g., by better models for reward and attention functions, and better word-embedding representations of text.

2.5 Additional Details

Visualization of Learned Reward and Value

In Figure 2.4 we plot the learned reward and value function for the gridworld task. The learned reward is very negative at obstacles, very positive at goal, and a slightly negative constant otherwise. The resulting value function has a peak at the goal, and a gradient pointing towards a direction to the goal around obstacles. This plot clearly shows that the VI block learned a useful planning computation.

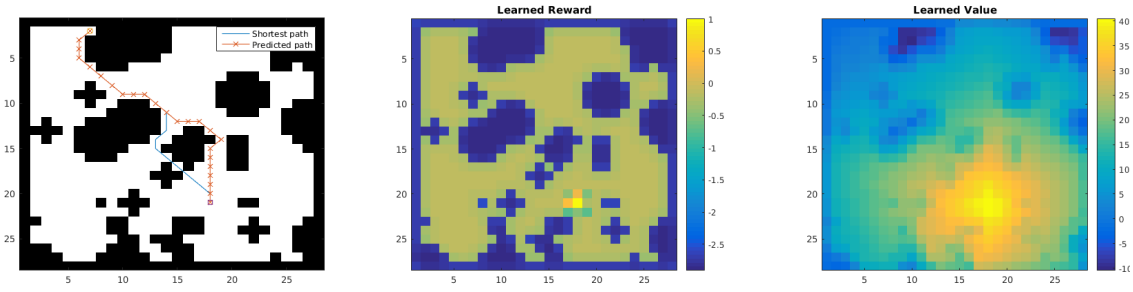


Figure 2.4: Visualization of learned reward and value function. Left: a sample domain. Center: learned reward f_R for this domain. Right: resulting value function (in VI block) for this domain.

Weight Sharing

The VINs have an effective depth of K, which is larger than the depth of the reactive policies. One may wonder, whether any deep enough network would learn to plan. In principle, a CNN

or FCN of depth K has the potential to perform the same computation as a VIN. However, it has much more parameters, requiring much more training data. We evaluate this by untying the weights in the K recurrent layers in the VIN. Our results, in Table 2.2 show that untying the weights degrades performance, with a stronger effect for smaller sizes of training data.

Training data	VIN			VIN Untied Weights		
	Pred. loss	Succ. rate	Traj. diff.	Pred. loss	Succ. rate	Traj. diff.
20%	0.06	98.2%	0.106	0.09	91.9%	0.094
50%	0.05	99.4%	0.018	0.07	95.2%	0.078
100%	0.05	99.3%	0.089	0.05	95.6%	0.068

Table 2.2: Performance on 16×16 grid-world domain. Evaluation of the effect of VI module shared weights relative to data size.

Technical Details for Experiments

We report the full technical details used for training our networks.

Grid-world Domain

Our training set consists of $N_i = 5000$ random grid-world instances, with $N_t = 7$ shortest-path trajectories (calculated using an optimal planning algorithm) from a random start-state to a random goal-state for each instance; a total of $N_i \times N_t$ trajectories. For each state $s = (i, j)$ in each trajectory, we produce a $(2 \times m \times n)$-sized observation image s_{image}. The first channel of s_{image} encodes the obstacle presence (1 for obstacle, 0 otherwise), while the second channel encodes the goal position (1 at the goal, 0 otherwise). The full observation vector is $\phi(s) = [s, s_{\text{image}}]$. In addition, for each state we produce a label a that encodes the action (one of 8 directions) that an optimal shortest-path policy would take in that state.

We design a VIN for this task as follows. The state space $\bar{S}$ was chosen to be a $m \times n$ grid-world, similar to the true state space S.[4] The reward $\bar{R}$ in this space can be represented by an $m \times n$ map, and we chose the reward mapping f_R to be a CNN with s_{image} as its input, one layer with 150 kernels of size 3×3, and a second layer with one 3×3 filter to output $\bar{R}$. Thus, f_R maps the image of obstacles and goal to a 'reward image'. The transitions $\bar{P}$ were defined as 3×3 convolution kernels in the VI block, and exploit the fact that transitions in the grid-world are local. Note that the transitions defined this way do not depend on

[4]For a particular configuration of obstacles, the true grid-world domain can be captured by a $m \times n$ state space with the obstacles encoded in the MDP transitions, as in our notation. For a general obstacle configuration, the obstacle positions have to also be encoded in the state. The VIN was able to learn a policy for a general obstacle configuration by planning in a $m \times n$ state space by also taking into account the observation of the map.

the state $\bar{s}$. Interestingly, we shall see that the network learned rewards and transitions that nevertheless enable it to successfully plan in this task. For the attention module, since there is a one-to-one mapping between the agent position in S and in $\bar{S}$, we chose a trivial approach that selects the $\bar{Q}$ values in the VI block for the state in the real MDP s, i.e., $\psi(s) = \bar{Q}(s, \cdot)$. The final reactive policy is a fully connected softmax output layer with weights W, $\pi_{\text{re}}(\cdot|\psi(s)) \propto \exp\left(W^\top \psi(s)\right)$.

We trained several neural-network policies based on a multi-class logistic regression loss function using stochastic gradient descent, with an RMSProp step size [Tieleman and Hinton, 2012], implemented in the Theano [Al-Rfou et al., 2016] library.

We compare the policies:

VIN network We used the VIN model of Section 2.3 as described above, with 10 channels for the q layer in the VI block. The recurrence K was set relative to the problem size: $K = 10$ for 8×8 domains, $K = 20$ for 16×16 domains, and $K = 36$ for 28×28 domains. The guideline for choosing these values was to keep the network small while guaranteeing that goal information can flow to every state in the map.

CNN network: We devised a CNN-based reactive policy inspired by the recent impressive results of DQN [Mnih et al., 2015], with 5 convolution layers with $[50, 50, 100, 100, 100]$ kernels of size 3×3, and 2×2 max-pooling after the first and third layers. The final layer is fully connected, and maps to a softmax over actions. To represent the current state, we added to s_{image} a channel that encodes the current position (1 at the current state, 0 otherwise).

Fully Convolutional Network (FCN): The problem setting for this domain is similar to semantic segmentation [Long et al., 2015], in which each pixel in the image is assigned a semantic label (the action in our case). We therefore devised an FCN inspired by a state-of-the-art semantic segmentation algorithm [Long et al., 2015], with 3 convolution layers, where the first layer has a filter that spans the whole image, to properly convey information from the goal to every other state. The first convolution layer has 150 filters of size $(2m-1) \times (2n-1)$, which span the whole image and can convey information about the goal to every pixel. The second layer has 150 filters of size 1×1, and the third layer has 10 filters of size 1×1, to produce an output sized $10 \times m \times n$, similarly to the $\bar{Q}$ layer in our VIN. Similarly to the attention mechanism in the VIN, the values that correspond to the current state (pixel) are passed to a fully connected softmax output layer.

WebNav

"WebNav" [Nogueira and Cho, 2016] is a recently proposed goal-driven web navigation benchmark. In WebNav, web pages and links from some website form a directed graph $G(S, E)$. The agent is presented with a query text, which consists of N_q sentences from a target page at most N_h hops away from the starting page. The goal for the agent is to navigate to that target page from the starting page via clicking at most N_n links per page. Here, we choose $N_h = N_q = N_n = 4$. In [Nogueira and Cho, 2016], the agent receives

a reward of 1 when reaching the target page via any path no longer than 10 hops. For evaluation convenience, in our experiment, the agent can receive a reward only if it reaches the destination via the *shortest path*, which makes the task much harder. We measure the top-1 and top-4 prediction accuracy as well as the average reward for the baseline [Nogueira and Cho, 2016] and our VIN model.

For every page s, the valid transitions are $A_s = \{s' : (s, s') \in E\}$.

For every web page s and every query text q, we utilize the bag-of-words model with pretrained word embedding provided by [Nogueira and Cho, 2016] to produce feature vectors $\phi(s)$ and $\phi(q)$. The agent should choose at most N_n valid actions from $A_s = \{s' : (s, s') \in E\}$ based on the current s and q.

The baseline method of [Nogueira and Cho, 2016] uses a single tanh-layer neural net parametrized by W to compute a hidden vector h: $h(s, q) = \tanh\left(W \begin{bmatrix} \phi(s) \\ \phi(q) \end{bmatrix}\right)$. The final baseline policy is computed via $\pi_{\text{bsl}}(s'|s, q) \propto \exp\left(h(s, q)^\top \phi(s')\right)$ for $s' \in A_s$.

We design a VIN for this task as follows. We firstly selected a smaller website as the approximate graph $\bar{G}(\bar{S}, \bar{E})$, and choose $\bar{S}$ as the states in VI. For query q and a page $\bar{s}$ in $\bar{S}$, we compute the reward $\bar{R}(\bar{s})$ by $f_R(\bar{s}|q) = \tanh\left((W_R\phi(q) + b_R)^\top \phi(\bar{s})\right)$ with parameters W_R (diagonal matrix) and b_R (vector). For transition, since the graph remains unchanged, $\bar{P}$ is fixed. For the attention module $\Pi(\bar{V}^\star, s)$, we compute it by $\Pi(\bar{V}^\star, s) = \sum_{\bar{s}\in\bar{S}} \text{sigmoid}\left((W_\Pi\phi(s) + b_\Pi)^\top \phi(\bar{s})\right)\bar{V}^\star(\bar{s})$, where W_Π and b_Π are parameters and W_Π is diagonal. Moreover, we compute the coefficient γ based on the query q and the state s using a tanh-layer neural net parametrized by W_γ: $\gamma(s, q) = \tanh\left(W_\gamma \begin{bmatrix} \phi(s) \\ \phi(q) \end{bmatrix}\right)$. Finally, we combine the VI module and the baseline method as our VIN model by simply adding the outputs from these two networks together.

In addition to the experiments reported in the main text, we performed experiments on the full wikipedia, using 'wikipedia for schools' as the graph for VIN planning. We report our preliminary results here.

Full wikipedia website: The full wikipedia dataset consists 779169 training queries (3 million training samples) and 20004 testing queries (76664 testing samples) over 4.8 million pages with maximum 300 links per page.

We use the whole WikiSchool website as our approximate graph and set $K = 4$. In VIN, to accelerate training, we firstly only train the VI module with $K = 0$. Then, we fix $\bar{R}$ obtained in the $K = 0$ case and jointly train the whole model with $K = 4$. The results are shown in Tab. 2.3

VIN achieves 1.5% better prediction accuracy than the baseline. Interestingly, with only 1.5% prediction accuracy enhancement, VIN achieves 2.5% better success rate than the baseline: note that the agent can only success when making 4 consecutive correct predictions. This indicates the VI does provide useful high-level planning information.

Network	Top-1 Test Err.	Top-4 Test Err.	Avg. Reward
BSL	52.019%	24.424%	0.27779
VIN	50.562%	26.055%	0.30389

Table 2.3: Performance on the full wikipedia dataset.

2.6 Summary

The introduction of powerful and scalable RL methods has opened up a range of new problems for deep learning. However, few recent works investigate policy architectures that are specifically tailored for planning under uncertainty, and current RL theory and benchmarks rarely investigate the generalization properties of a trained policy [Sutton and Barto, 1998, Mnih et al., 2015, Duan et al., 2016a]. This chapter takes a step in this direction, by exploring better generalizing policy representations.

Our VIN policies learn an approximate planning computation relevant for solving the task, and we have shown that such a computation leads to better generalization in different tasks from simple gridworlds to navigation of Wikipedia links. For future work, it is plausible to learn different planning computations, e.g., based on simulation [Guo et al., 2014], or optimal linear control [Watter et al., 2015], and combine them with reactive policies, to potentially develop RL solutions for a wider range of domains.

Part II

Visual Semantic Generalization for Embodied Agents

Chapter 3

House3D: an Environment for Building Generalizable Agents

Teaching an agent to navigate in an unseen 3D environment is a challenging task, even in the event of simulated environments. To generalize to unseen environments, an agent needs to be robust to low-level variations (e.g. color, texture, object changes), and also high-level variations (e.g. layout changes of the environment). To improve overall generalization, all types of variations in the environment have to be taken under consideration via different level of data augmentation steps. To this end, we propose *House3D*, a rich, extensible and efficient environment that contains 45,622 human-designed 3D scenes of visually realistic houses, ranging from single-room studios to multi-storied houses, equipped with a diverse set of fully labeled 3D objects, textures and scene layouts, based on the SUNCG dataset [Song et al., 2017b]. The diversity in House3D opens the door towards scene-level augmentation, while the label-rich nature of House3D enables us to inject pixel- & task-level augmentations such as domain randomization [Tobin et al., 2017] and multi-task training. Using a subset of houses in House3D, we show that reinforcement learning agents trained with an enhancement of different levels of augmentations perform much better in unseen environments than our baselines with raw RGB input by over 8% in terms of navigation success rate. House3D is publicly available at http://github.com/facebookresearch/House3D.

3.1 Motivation

Recently, deep reinforcement learning has shown its strength on multiple games, such as Atari [Mnih et al., 2015] and Go [Silver et al., 2016], vastly overpowering human performance. Via the various reinforcement learning frameworks, different aspects of intelligence can be learned, including 3D understanding (DeepMind Lab [Beattie et al., 2016] and Malmo [Johnson et al., 2016]), real-time strategy decision (TorchCraft [Synnaeve et al., 2016] and ELF [Tian et al., 2017]), fast reaction (Atari [Bellemare et al., 2013]), long-term planning (Go, Chess), language and communications (ParlAI [Miller et al., 2017] and [Das et al., 2017]).

A prominent issue in reinforcement learning is *generalizability*. Commonly, agents trained on a specific environment and for a specific task become highly specialized and fail to perform well on new environments. In the past, there have been efforts to address this issue. In particular, pixel-level variations are applied to the observation signals in order to increase the agent's robustness to unseen environments [Beattie et al., 2016, Higgins et al., 2017, Tobin et al., 2017]. Parametrized environments with varying levels of difficulty are used to yield scene variations but with similar visual observations [Pathak et al., 2017]. Transfer learning is applied to similar tasks but with different rewards [Finn et al., 2017b].

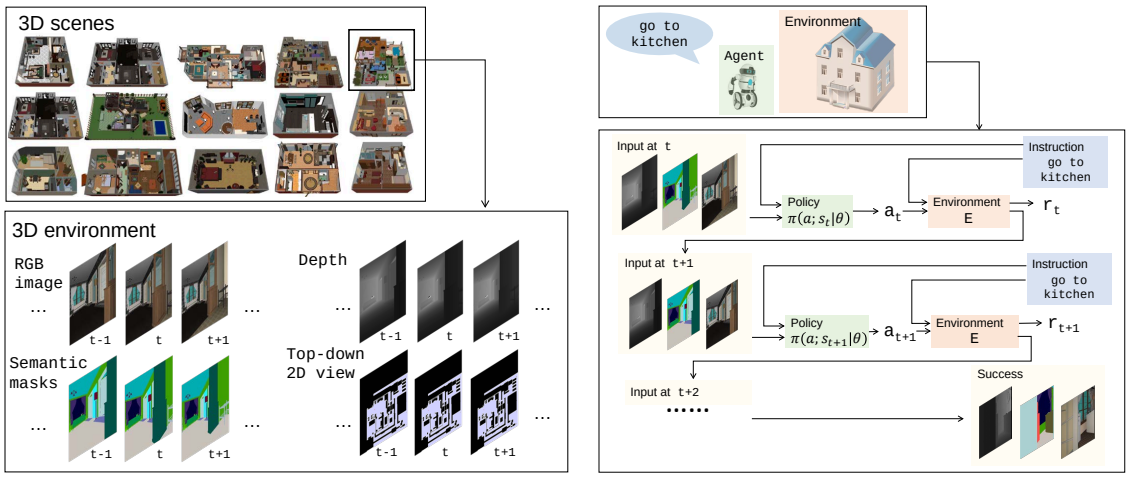
Nevertheless, the aforementioned techniques study the problem in simplified environments which lack the diversity, richness and perception challenges of the real world. To this end, we propose a substantially more diverse environment, *House3D*, to train and test our agents. House3D is a virtual 3D environment consisting of thousands of indoor scenes equipped with a diverse set of scene types, layouts and objects. An overview of House3D is shown in Figure 3.1a. House3D leverages the SUNCG dataset [Song et al., 2017b] which contains 45K *human-designed* real-world 3D house models, ranging from single studios to houses with gardens, in which objects are fully labeled with categories. We convert the SUNCG dataset to an *environment*, House3D, which is efficient and extensible for various tasks. In House3D, an agent can freely explore the space while perceiving a large number of objects under various visual appearances.

Based on House3D, we design a task called *RoomNav*: an agent starts at a random location in a house and is asked to navigate to a destination specified by a high-level semantic concept (e.g. *kitchen*), following simple rules (e.g. no object penetration), as shown in Figure 3.1b. We use gated-CNN and gated-LSTM policies trained with standard deep reinforcement learning methods, i.e. A3C [Mnih et al., 2016] and DDPG [Lillicrap et al., 2015], and report success rate on unseen environments over 5 concepts. We show that in order to achieve strong generalization capability, **all-levels of augmentations are needed**: pixel-level augmentation by domain randomization [Tobin et al., 2017] enhances the agent's robustness to color variations; object-level augmentation forces the agent to learn multiple concepts (20 in number) simultaneously, and scene-level augmentation, where a diverse set of environments is used, enforce generalizability across diverse scenes, mitigating overfitting to particular scenes. Our final gated-LSTM agent achieves a success rate of 35.8% on 50 unseen environments, 10% better than the baseline method (25.7%).

The remaining of the chapter is structured as follows. Section 3.1 summarizes relevant work. Section 3.2 describes our environment, House3D, in detail and section 3.3 describes the task, RoomNav. Section 3.4 describes our gated models and the applied algorithms to tackle RoomNav. Finally, experimental results are shown in Section 3.5 with additional details in Section 3.6.

Related Work

Environments: Table 3.1 shows the comparison between House3D and most relevant prior works. There are other simulated environments which focus on different domains, such as



(a) House3D environment (b) RoomNav task

Figure 3.1: An overview of House3D environment and RoomNav task. **(a)** We build an efficient and interactive environment upon the SUNCG dataset [Song et al., 2017b] that contains 45K diverse indoor scenes, ranging from studios to two-storied houses with swimming pools and fitness rooms. All 3D objects are fully labeled into over 80 categories. Observations of agents in the environment have multiple modalities, including RGB images, Depth, Segmentation masks (from object category), top-down 2D view, etc. **(b)** We focus on the task of targeted navigation. Given a high-level description of a room concept, the agent explores the environment to reach the target room.

Environment	3D	Realistic	Large-scale	Fast	Customizable
Atari [Bellemare et al., 2013]				•	
OpenAI Universe [Shi et al., 2017]		•	•		•
Malmo [Johnson et al., 2016]	•		•	•	•
DeepMind Lab [Beattie et al., 2016]	•			•	•
VizDoom [Kempka et al., 2016]	•			•	•
AI2-THOR [Zhu et al., 2017b]	•	•		•	
Stanford2D-3D [Armeni et al., 2016]	•	•		•	
Matterport3D [Chang et al., 2017]	•	•	•	•	
House3D	•	•	•	•	•

Table 3.1: A summary of popular environments. The attributes include **3D**: 3D nature of the rendered objects, **Realistic**: resemblance to the real-world, **Large-scale**: a large set of environments, **Fast**: fast rendering speed and **Customizable**: flexibility to be customized to other applications.

OpenAI Gym [Brockman et al., 2016], ParlAI [Miller et al., 2017] for language communication as well as some strategic game environments [Synnaeve et al., 2016, Tian et al., 2017, Vinyals et al., 2017], etc. Most of these environments are pertinent to one particular aspect of intelligence, such as dialogue or a single type of game, which makes it hard to facilitate the study of more comprehensive problems. On the contrary, we focus on building a platform that intersects with multiple research directions, such as object and scene understanding, 3D navigation, embodied question answering [Das et al., 2018a], while allowing users to customize the level of complexity to their needs.

We build on SUNCG [Song et al., 2017b], a dataset that consists of thousands of diverse synthetic indoor scenes equipped with a variety of objects and layouts. Its visual diversity and rich content opens the path to the study of semantic generalization for reinforcement learning agents. Our platform decouples high-performance rendering from data I/O, and thus can use other publicly available 3D scene datasets as well. This includes Al2-THOR [Zhu et al., 2017b], SceneNet RGB-D [McCormac et al., 2017], Stanford 3D [Armeni et al., 2016], Matterport 3D [Chang et al., 2017] and so on.

Concurrent works [Brodeur et al., 2017, Savva et al., 2017] also introduce similar platforms as House3D, indicating the interest for large-scale interactive and realistic 3D environments.

3D Navigation: There has been a prominent line of work on the task of navigation in real 3D scenes [Leonard and Durrant-Whyte, 1992]. Classical approaches decompose the task into two subtasks by building a 3D map of the scene using SLAM and then planning in this map [Fox et al., 2005]. More recently, end-to-end learning methods were introduced to predict robotic actions from raw pixel data [Levine et al., 2016]. Some of the most recent works on navigation show the effectiveness of end-to-end learning. [Gupta et al., 2017b] learn to navigate via mapping and planning using shortest path supervision. [Sadeghi and Levine, 2017] teach an agent to fly using solely simulated data and deploy it in the real world. [Gandhi et al., 2017] collect a dataset of drones crashing into objects and train self-supervised agents on this data to avoid obstacles.

A number of recent works also use deep reinforcement learning for navigation in simulated 3D scenes. [Mirowski et al., 2017, Jaderberg et al., 2017] improve an agent's navigation ability in mazes by introducing auxiliary tasks. [Parisotto and Salakhutdinov, 2018] propose a new architecture which stores information of the environment on a 2D map. [Karl Moritz Hermann and PhilBlunsom, 2017] focus on the task of language grounding by navigating simple 3D scenes. However, these works only evaluate the agent's generalization ability on pixel-level variations or small mazes. We argue that a much richer environment is crucial for evaluating semantic-level generalization.

Gated Modules: In our work, we focus on the task of RoomNav, where the goal is communicated to the agent as a high-level instruction selected from a set of predefined concepts. To modulate the behavior of the agent in RoomNav, we encode the instruction as an embedding vector which gates the visual signal. The idea of gated attention has been used in the past for language grounding [Chaplot et al., 2018], and transfer learning by language grounding [Narasimhan et al., 2018]. Similar to those works, we use concept grounding as an attention mechanism. We believe that our gated reinforcement learning models serve as

a strong baseline for the task of semantic based navigation in House3D. Furthermore, our empirical results allow us to draw conclusions on the models' efficacy when training agents in a large-scale, diverse dataset with an emphasis on generalization.

Generalization: There is a recent trend in reinforcement learning focusing on the problem of generalization, ranging from learning to plan [Tamar et al., 2016], meta-learning [Duan et al., 2016b, Finn et al., 2017a] to zero-shot learning [Andreas et al., 2017, Oh et al., 2017, Higgins et al., 2017]. However, these works either focus on over-simplified tasks or test on environments which are only slightly varied from the training ones. In contrast, we use a more diverse set of environments, each containing visually and structurally different observations, and show that the agent can work well in unseen scenes.

In this work, we show improved generalization performance in complex 3D scenes when using depth and segmentation masks on top of the raw visual input. This observation is similar to other works which use a diverse set of input modalities [Mirowski et al., 2017, Tai and Liu, 2016]. Our result suggests that it can be possible to decouple real-world robotics from recognition via a vision API provided by an object detection or semantic segmentation system trained on the targeted real scenes. This opens the door towards bridging the gap between simulated environment and real-world [Tobin et al., 2017, Rusu et al., 2017, Christiano et al., 2016].

3.2 House3D: An Extensible Environment of 45K 3D Houses

We propose House3D, an environment which closely resembles the real world and is rich in content and structure. An overview of House3D is shown in Figure 3.1a. House3D is developed to provide an efficient and flexible environment of thousands of indoor scenes and facilitates a variety of tasks, e.g. navigation, visual understanding, language grounding, concept learning etc. The environment along with a python API for easy use is available at `http://github.com/facebookresearch/House3D`.

Dataset

The 3D scenes in House3D are sourced from the SUNCG dataset [Song et al., 2017b], which consists of 45,622 human-designed 3D scenes ranging from single-room studios to multi-floor houses. The SUNCG dataset was designed to encourage research on large-scale 3D object recognition problems and thus carries a variety of objects, scene layouts and structures. On average, there are 8.9 rooms and 1.3 floors per scene There is a diverse set of room and object types in each scene. In total, there are over 20 different room types, such as bedroom, living room, kitchen, bathroom etc., with over 80 different object categories. In total, the SUNCG dataset contains 404,508 different rooms and 5,697,217 object instances drawn from 2644 unique object meshes.

Annotations

Each scene in SUNCG is fully annotated with 3D coordinates and its room and object types (e.g. bedroom, shoe cabinet, etc). This allows for a detailed mapping from each 3D location to an object instance (or None at free space) and the room type.

At every time step an agent has access to the following signals: a) the visual RGB signal of its current first person view, b) semantic/instance segmentation masks for all the objects visible in its current view, and c) depth information. For different tasks, these signals might serve for different purposes, e.g., as a feature plane or an auxiliary target. Based on the existing annotations, House3D offers more information, e.g., top-down 2D occupancy maps, connectivity analysis and shortest paths between two points.

Renderer

To build a realistic 3D environment, we develop a renderer for the SUNCG scenes. The renderer is based on OpenGL, it can run on both Linux and MacOS, and provides RGB images, semantic segmentation masks, instance segmentation masks and depth maps.

As highlighted above, the environment needs to be *efficient* in order to be used for large-scale reinforcement learning. On a NVIDIA Tesla M40 GPU, our implementation can render 120×90-sized frames at over 600 fps, while multiple renderers can run in parallel on one or more GPUs. When rendering multiple houses simultaneously, one M40 GPU can be fully utilized to render at a total of 1800 fps. The default simple physics adds a small overhead to the rendering. The high throughput of our implementation enables efficient learning for a variety of interactive tasks, such as on-policy reinforcement learning.

Interaction

In House3D, an agent can live in any location within a 3D scene, as long as it does not collide with object instances (including walls) within a small range, i.e. robot's radius. Doors, gates and arches are considered passage ways, meaning that an agent can walk through those structures freely. These default design choices add negligible run-time overhead. Note that more complex interaction rules can be incorporated (e.g. manipulation) within House3D using our flexible API, which we leave for future work.

3.3 RoomNav: A Benchmark Task for Concept-Driven Navigation

Consider the task of concept-driven navigation as shown in Figure 3.1b. A human may give a high level instruction to the robot, for example, "Go to the *kitchen*", so that one can later ask the robot to turn on the oven. The robot needs to behave appropriately conditioned on the house it is located in and the goal, e.g. the semantic concept "kitchen". In addition, we

want the agent to *generalize*, i.e. to perform well in unseen environments, that is new houses with different layouts and furniture locations.

To study the aforementioned abilities of an agent, we develop a benchmark task, *Concept-Driven* Navigation (RoomNav), based on House3D. We define the goal to be of the form "go to `X`", where `X` denotes a pre-defined room type or object type, which is a semantic concept that an agent needs to interpret from a variety of scenes of distinct visual appearances. To ensure fast experimentation cycles, we perform experiments on a subset of House3D. We manually select 270 houses suitable for a navigation task and split them into a small set (20 houses), a large set (200 houses) and a test set (50 houses), where the test set is used to evaluate the generalization of the trained agents.

Task Formulation: Suppose we have a set of episodic environments $\mathcal{E} = \{E_1, .., E_n\}$ and a set of semantic concepts $\mathcal{I} = \{I_1, .., I_m\}$. During each episode, the agent is interacting with one environment $E \in \mathcal{E}$ and is given a concept $I \in \mathcal{I}$. In the beginning of an episode, the agent is randomly placed somewhere in E. At each time step t, the agent receives a visual signal X_t from E via its first person view sensor. Let $s_t = \{X_1, .., X_t, I\}$ denote the state of the agent at time t. The agent needs to propose an action a_t to navigate and rotate its sensor given s_t. The environment returns a reward signal r_t and terminates when the agent succeeds in finding the destination, or reaches a maximum number of steps.

The objective of this task is to learn an optimal policy $\pi(a_t|s_t, I)$ that leads to the target defined by I. We train the agent on a set $\mathcal{E}_{\text{train}}$. We evaluate the policy on a disjoint set of environments $\mathcal{E}_{\text{test}}$ ($\mathcal{E}_{\text{test}} \cap \mathcal{E}_{\text{train}} = \emptyset$). For more details see Section 3.6.

Environment Statistics: The selected 270 houses are manually verified for navigation; they are well connected, contain desired concepts, and are large enough for exploration. We split them into 3 disjoint sets, denoted by $\mathcal{E}_{small}$, $\mathcal{E}_{large}$ and $\mathcal{E}_{test}$ respectively. For the semantic concepts, we select the five most common room types: kitchen, living room, dining room, bedroom and bathroom. Note that this set can be extended to include objects or even subareas within rooms.

Observations: We utilize three different kinds of visual input signals for X_t, including (1) raw pixel values; (2) semantic segmentation mask of the pixel input; and (3) depth information, and experiment with different combinations of them. We encode each concept I as a one-hot vector representation.

Action Space: Similar to existing navigation works, we define a fixed set of actions, here 12 in number including different scales of rotations and movements. Due to the complexity of the indoor scenes, we also explore a continuous action space similar to [Lowe et al., 2017], which in effect allows the agent to move with different velocities. For more details see Section 3.6. In all cases, if the agent hits an obstacle it remains still.

Success Measure and Reward Function: To declare *success*, we want to ensure that the agent *identifies* the target room by its unique properties (e.g. presence of appropriate objects in the room such as pan and knives for kitchen and bed for bedroom) instead of merely reaching there by luck. An episode is considered successful if both of the following two criteria are satisfied: (1) the agent is located inside the target room; (2) the agent consecutively *sees* a designated object category associated with that target room type for at

least 2 time steps. We assume that an agent *sees* an object if there are at least 4% of pixels in X_t belonging to that object.

For the reward function, ideally two signals suffice to reflect the task requirement: (1) a collision penalty when hitting obstacles; and (2) a success reward when completing the task. However, these basic signals make it too difficult for an RL agent to learn, as the positive reward is too sparse. To provide additional supervision during training, we resort to an informative reward shaping: we compute the approximate shortest distance from the target room to each location in the house and adopt the difference of shortest distances between the agent's movement as an additional reward signal. Note that our ultimate goal is to learn a policy that could *generalize* to unseen houses. Our strong reward shaping supervises the agent at training and is *not available* to the agent at test time. We empirically observe that stronger reward shaping leads to better performances on both training and testing.

3.4 Gated-Attention Networks for Multi-Target Learning

The RoomNav task can be considered as a multi-target learning problem: the policy needs to condition on both the input s_t and the target concept I. For policy representations which incorporate the target I, we propose two baseline models with a gated-attention architecture, similar to [Dhingra et al., 2017] and [Chaplot et al., 2018]: a gated-CNN network for continuous actions and a gated-LSTM network for discrete actions. We train the gated-CNN policy using the deep deterministic policy gradient (DDPG) [Lillicrap et al., 2015], while the gated-LSTM policy is trained using the asynchronous advantage actor-critic algorithm (A3C) [Mnih et al., 2016].

DDPG with gated-CNN policy

Deep deterministic policy gradient

Suppose we have a deterministic policy $\mu(s_t|\theta)$ (actor) and the Q-function $Q(s_t, a|\theta)$ (critic) both parametrized by θ. DDPG optimizes the policy $\mu(s_t|\theta)$ by maximizing $L_\mu(\theta) = \mathbb{E}_{s_t}\left[Q(s_t, \mu(s_t|\theta)|\theta)\right]$, and updates the Q-function by minimizing $L_Q(\theta) = \mathbb{E}\left[(Q(s_t, a_t|\theta) - \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)\right.$

Here, we use a shared network for both actor and critic with the final loss function $L_{\text{DDPG}}(\theta) = -L_\mu(\theta) + \alpha_{\text{DDPG}} L_Q(\theta)$, where α_{DDPG} is a constant balancing the two objectives.

Gated-CNN for continuous policy

State Encoding: Given state s_t, we first stack the most recent k frames $X = [X_t, X_{t-1}, \ldots, X_{t-k+1}]$ channel-wise and apply a convolutional neural network to derive an image representation $x = f_{\text{cnn}}(X|\theta) \in \mathbb{R}^{d_X}$. We convert the target I into an embedding vector $y = f_{\text{embed}}(I|\theta) \in \mathbb{R}^{d_I}$. Subsequently, we apply a fusion module $M(x, y|\theta)$ to derive the final encoding $h_s = M(x, y|\theta)$.

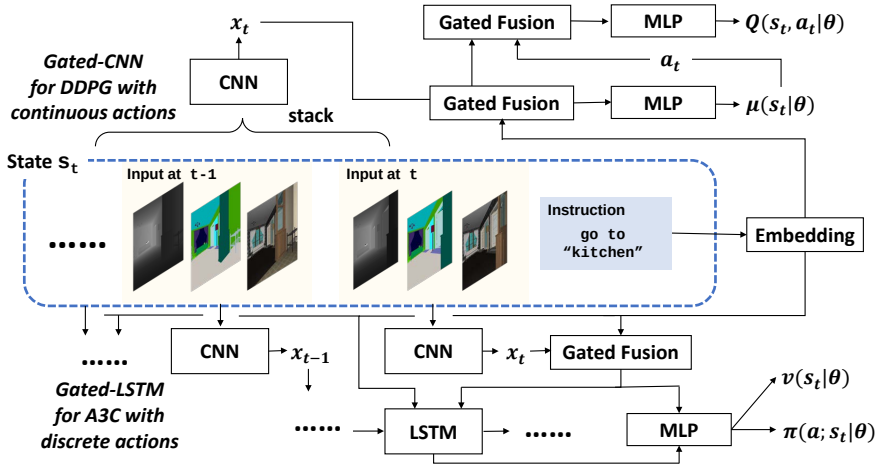


Figure 3.2: Overview of our proposed models. Bottom part demonstrates the gated-LSTM model for discrete action while the top part shows the gated-CNN model for continuous action. The "Gated Fusion" module denotes the gated-attention architecture.The input shows the modalities in SUNCG [Song et al., 2017b].

Gated-Attention for Feature Fusion: For the fusion module $M(x, y|\theta)$, the straightforward version is concatenation, namely $M_{\text{cat}}(x, y|\cdot) = [x, y]$. In our case, x is always a high-dimensional feature vector (i.e., image feature) while y is a simple low-dimensional conditioning vector (e.g., instruction). Thus, simple concatenation may result in optimization difficulties. For this reason, we propose to use a gated-attention mechanism. Suppose $x \in \mathbb{R}^{d_x}$ and $y \in \mathbb{R}^{d_y}$ where $d_y < d_x$. First, we transform y to $y' \in \mathbb{R}^{dX}$ via an MLP, namely $y' = f_{\text{mlp}}(y|\theta)$, and then perform a Hadamard (pointwise) product between x and sigmoid(y'), which leads to our final gated fusion module $M(x, y|\theta) = x \odot \text{sigmoid}(f_{\text{mlp}}(y|\theta))$. This gated fusion module could also be interpreted as an attention mechanism over the feature vector which could help better shape the feature representation.

Policy Representation: For the policy, we apply a MLP layer on the state representation h_s, followed by a softmax operator (for bounded velocity) to produce the continuous action. Moreover, in order to produce a stochastic policy for both better exploration and higher robustness, we apply the Gumbel-Softmax trick [Jang et al., 2017], resulting in the final policy $\mu(s_t|\theta) = \text{Gumbel-Softmax}(f_{\text{mlp}}(h_s|\theta))$. Note that since we add randomness to $\mu(s_t|\theta)$, our DDPG formulation can also be interpreted as the SVG(0) algorithm [Heess et al., 2015].

Q-function: The Q-function $Q(s, a)$ conditions on both state s and action a. We again apply a gated fusion module to the feature vector x and the action vector a to derive a hidden representation $h_Q = M(x, a|\theta)$. We eventually apply another MLP to h_Q to produce the final value $Q(s, a)$.

A model demonstration is shown in the top part of Fig. 3.2, where each block has its own parameters.

A3C with gated-LSTM policy

Asynchronous advantage actor-critic

Suppose we have a discrete policy $\pi(a; s|\theta)$ and a value function $v(s|\theta)$. A3C optimizes the policy by minimizing the loss function $L_{\text{pg}}(\theta) = -\mathbb{E}_{s_t,a_t,r_t}\left[\sum_{t=1}^{T}(R_t - v(s_t)) \log \pi(a_t; s_t|\theta)\right]$, where R_t is the discounted accumulative reward defined by $R_t = \sum_{i=0}^{T-t} \gamma^i r_{t+i} + v(s_{T+1})$. The value function is updated by minimizing the loss $L_v(\theta) = \mathbb{E}_{s_t,r_t}[(R_t - v(s_t))^2]$.

Finally the overall loss function for A3C is $L_{\text{A3C}}(\theta) = L_{\text{pg}}(\theta) + \alpha_{\text{A3C}} L_v(\theta)$ where α_{A3C} is a constant coefficient.

Gated-LSTM network for discrete policy

State Encoding: Given state s_t, we first apply a CNN module to extract image feature x_t for each input frame X_t. For the target, we apply a gated fusion module to derive a state representation $h_t = M(x_t, I|\theta)$ at each time step t. Then, we concatenate h_t with the target I and the result is fed into the LSTM module [Hochreiter and Schmidhuber, 1997] to obtain a sequence of LSTM outputs $\{o_t\}_t$, so that the LSTM module has direct access to the target other than the attended visual feature.

Policy and Value Function: For each time step t, we concatenate the state vector h_t with the output of the LSTM o_t to obtain a joint hidden vector $h_{\text{joint}} = [h_t, o_t]$. Then we apply two MLPs to h_{joint} to obtain the policy distribution $\pi(a; s_t|\theta)$ as well as the value function $v(s_t|\theta)$.

A visualization of the model is in the bottom part of Fig. 3.2. The parameters of CNN modules are shared across time.

3.5 Experiments

We report experimental results for our models on the task of RoomNav. We first compare models with discrete and continuous action spaces with different input modalities. Then we explain our observations and show that techniques targeting different levels of augmentation improve the success rate of navigation in the test set. Moreover, these techniques are complementary to each other.

Setup. We train our baseline models on multiple experimental settings. We use two training datasets. The small set $\mathcal{E}_{\text{small}}$ contains 20 houses and the large set $\mathcal{E}_{\text{large}}$ contains 200 houses. A held-out dataset $\mathcal{E}_{\text{test}}$ is used for test, which contains 50 houses.

We mainly focus on success rate on the test set, i.e, how the agent generalizes. For reference, we also report the training performance. The agent fails if it failed to find the concept within 100 steps[1]. All success rate evaluations use a fixed random seed for a fair

[1]This is enough for success evaluation. The average number of steps for success runs in every setting is less than 45, which is much smaller than 100. Refer to Section 3.6 for details.

comparison. For each model, we run 2000 evaluation episodes on $\mathcal{E}_{\text{small}}$ and $\mathcal{E}_{\text{test}}$, and 5000 evaluation episodes on $\mathcal{E}_{\text{large}}$ to measure overall success rates.

We use `gated-CNN` and `gated-LSTM` to denote the networks with gated-attention, and `concat-CNN` and `concat-LSTM` for models with simple concatenation. We also experiment with different visual signals to the agents, including RGB image (RGB Only), RGB image with depth information (RGB+Depth) and semantics mask with depth information (Mask+Depth). The input image resolution is 120×90 to preserve image details.

During each simulated episode, we randomly select a house from the environment set and randomly pick an applicable target from the house to instruct the agent. During training, we add an entropy bonus term for both models[2] in addition to the original loss function. For evaluation, we keep the final model for DDPG due to its stable learning curve, while for A3C, we take the model with the highest training success rate. We use Pytorch [Paszke et al., 2019] and Adam [Kingma and Ba, 2014]. See Section 3.6 for more experiment details.

Baselines: Models with RGB Signals on $\mathcal{E}_{\text{small}}$

As shown in the bottom part of Fig. 3.3a, on $\mathcal{E}_{\text{small}}$, the test success rate for models trained on RGB features is unsatisfactory. We observe obvious overfitting behavior: the test performance is drastically worse than training. In particular, the gated-LSTM models achieve even lower success rate than concat-LSTM models, despite the fact that they have much better training performance. In this case, the learning algorithm picks up spurious color patterns in the environments as the guidance towards the goal, which is inapplicable to unseen environments.

In both training and test, we find that depth information improves the performance thus we use it in the following experiments and omit *Depth* for conciseness.

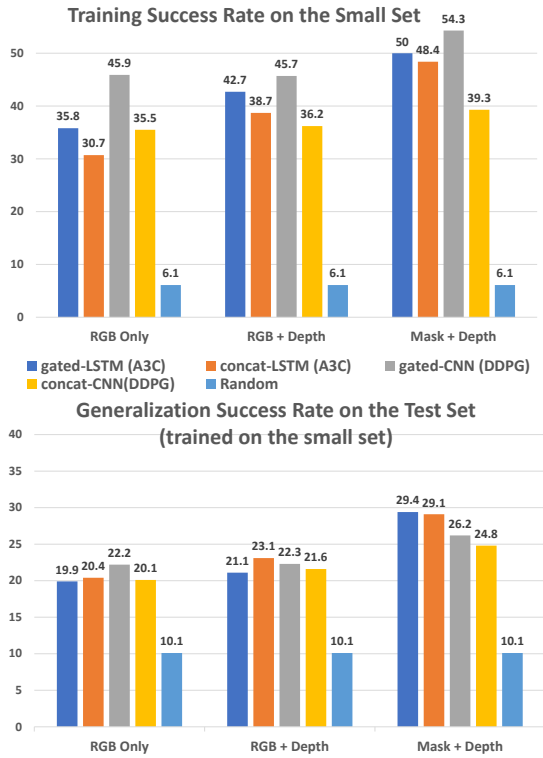
Techniques for Different Levels of Augmentation

Augmentation is a standard technique to improve generalization. However, for complicated tasks, augmentation needs to be taken care at different levels. In this section, we categorize augmentation techniques into 3 levels: (1) pixel-level augmentation: changing the colors and textures; (2) task-level augmentation: joint learning for multiple tasks; (3) scene-level augmentation: training on more environments. We analyze the generalization performance with all techniques and conclude that these techniques are complementary and that the best test performance is obtained by combining these techniques together.

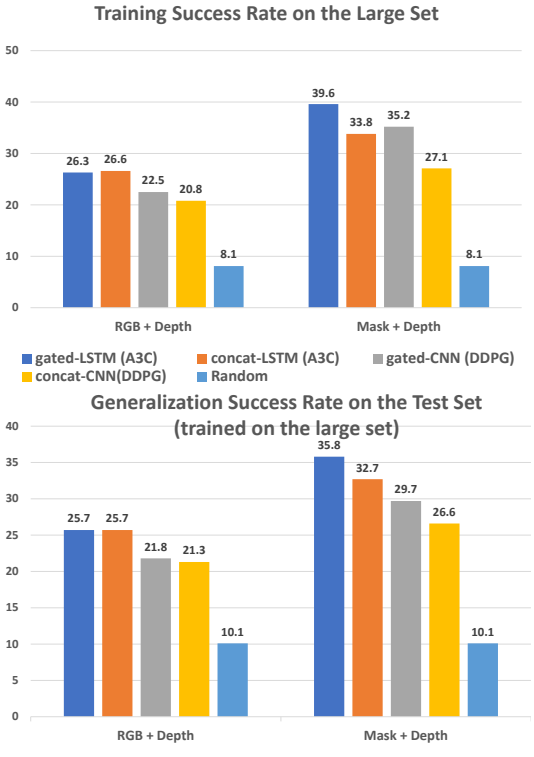
Pixel-level Augmentation: We use domain randomization [Tobin et al., 2017], by reassigning each object in the scene a random color but keeping the textures. This breaks the spurious color correlations and pushes the agent to learn a better representation.

We explore domain randomization by generating an additional 180 houses with random object coloring from $\mathcal{E}_{\text{small}}$, which leads to a total of 200 houses. We evaluate the test success rate of various models under different training settings, e.g., RGB, RGB with domain

[2]For DDPG, we simply use the entropy of the softmax output.



(a) Training (top) and test (bottom) results on small set



(b) Training (top) and test (bottom) results on large set

Figure 3.3: **Overall** performance of various models trained on (a) $\mathcal{E}_{\text{small}}$ (20 houses) with different input signals: RGB Only, RGB+Depth and Mask+Depth; (b) $\mathcal{E}_{\text{large}}$ (200 houses) with input signals: RGB+Depth and Mask+Depth. In each group, the bars from left to right correspond to gated-LSTM, concat-LSTM, gated-CNN, concat-CNN and random policy respectively.

randomization (D.R.) or mask signal. The results are shown in Fig. 3.4. Interestingly, we noticed that domain randomization yields very similar performance as mask signal on $\mathcal{E}_{\text{small}}$.

One shortcoming of domain randomization is that it requires substantially more training samples and thus suffers from high sample complexity. Thanks to the rich labels in House3D, we instead could use segmentation mask as an input feature plane, which encodes semantic information and is independent of the object color. This helps train generalizable agent with much fewer training samples. On the other hand, an agent trained with domain randomization can operate with RGB input only, without segmentation mask output from a vision subsystem. In the current context, we simply assume adopting segmentation mask input as the technique for pixel-level augmentation.

Task-level Augmentation: We explore task-level augmentation by adding related auxiliary targets during training (Fig. 3.5). Specifically, in addition to the 5 room types as auxiliary targets, we selected 15 object concepts (e.g., chair, table, cabinet, etc. See a full list

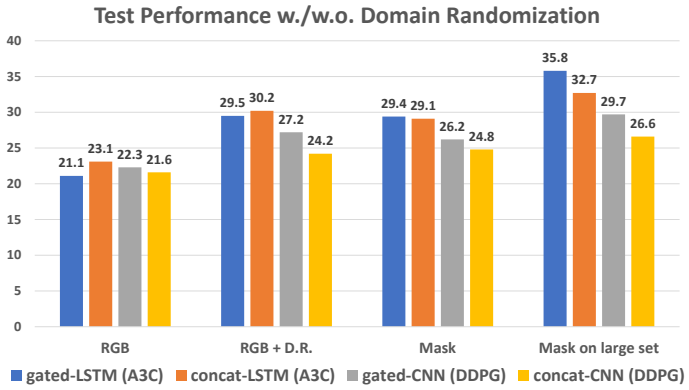


Figure 3.4: **Pixel-level Augmentation:** Test performances of various models trained with different input signals, including RGB+Depth on $\mathcal{E}_{\text{small}}$, RGB with Domain Randomization on $\mathcal{E}_{\text{small}}$, Mask+Depth on $\mathcal{E}_{\text{small}}$, Mask+Depth on $\mathcal{E}_{\text{large}}$. In each group, the bars represent gated-LSTM, concat-LSTM, gated-CNN and concat-CNN from left to right.

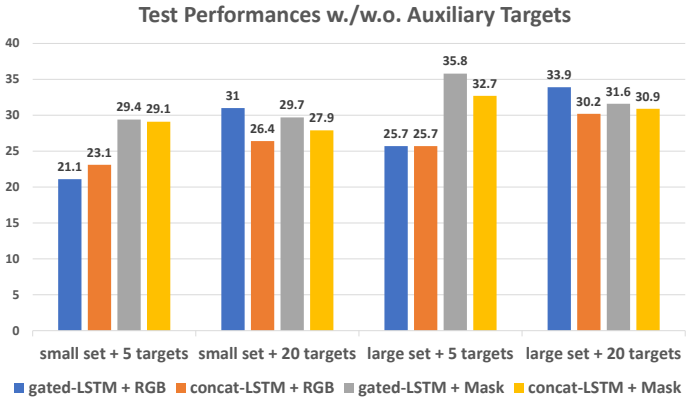


Figure 3.5: **Task-Level Augmentation:** Test performances of LSTM models trained with and without auxiliary targets on both $\mathcal{E}_{\text{small}}$ and $\mathcal{E}_{\text{large}}$. In each group, the bars represent gated-LSTM + RGB, concat-LSTM + RGB, gated-LSTM + Mask and concat-LSTM + Mask from left to right.

of object concepts in Section 3.6.). We train A3C agents with different input signals on $\mathcal{E}_{\text{small}}$ and evaluate their test performances.

We found that auxiliary targets significantly reduce overfitting and increases the generalizability of models with RGB inputs. Because of this effect, gated attention model, which has high model capacity, becomes much more effective on RGB signal when trained with more targets. On the other hand, with mask input, the agent does not need to learn to differentiate the objects, therefore auxiliary targets do not help that much for more complicated models like gated attention models.

Scene-level Augmentation: We could further boost the generalization performance

by augmenting the training set with more diverse set of houses, i.e, $\mathcal{E}_{\text{large}}$ that contain 200 different houses. This is also a benefit from House3D.

For visual signals, we focus on feature combinations like "RGB + Depth" and "Mask + Depth". Note that for training efficiency, segmentation mask is a surrogate feature to approximate "RGB + domain randomization" as it shows similar results in the small set. Both train and test results are summarized in Fig. 3.3b.

On a semantically diverse dataset $\mathcal{E}_{\text{large}}$, the overfitting issue is largely resolved. We see drops in the training performance and improve on the generalization. After training on a large number of environments, every model now has a much smaller gap between its training and test performance. This is in particularly true for the models using RGB signal, which suffers from overfitting issues on $\mathcal{E}_{\text{small}}$. Notably, on large dataset, LSTM models generally perform better than CNN models due to its high model capacity.

In addition, similar behavior was also observed during our experiments with techniques for pixel-level augmentation (Fig. 3.4) and task-level augmentation (Fig. 3.5). In all the experiments, all the models consistently achieves better generalization performances when trained on $\mathcal{E}_{\text{large}}$, which again emphasizes the benefits of House3D.

The overall best success rate is achieved by gated-attention architecture with semantic signals. It is better than both RGB channels by over 8% and the counterpart trained on $\mathcal{E}_{\text{small}}$ in terms of generalization metric. This means that pixel-level augmentation (e.g., domain randomization and/or segmentation mask) and scene-level augmentation (e.g., using diverse dataset) can improve the performance. Moreover, their effects are complementary.

A diverse environment like $\mathcal{E}_{\text{large}}$ also enables the model of larger capacity to work better. For example, LSTMs considerably outperform the simpler reactive models, i.e., CNNs with recent 5 frames as state input. We believe this is due to the larger scale and the high complexity of the training set, which makes it almost impossible for an agent to "remember" the optimal actions for every scenario. Instead, an agent needs to develop high-level abstractions (e.g., high-level exploration strategy, memory, etc). These are helpful induction biases that could lead to a more generalizable model.

Lastly, we also analyze the detailed success rate with respect to each target room in Section 3.6.

3.6 Additional Details

RoomNav Task Details

Statistics of Selected House Sets

We show the statistics of the selected three set of houses in Table 3.2.

In addition to these 5 houses, we also pick another 15 object concepts in our mid-level generalization experiment as auxiliary targets. The object concepts are: *shower*, *sofa*, *toilet*, *bed*, *plant*, *television*, *table-and-chair*, *chair*, *table*, *kitchen-set*, *bathtub*, *vehicle*, *pool*, *kitchen-cabinet*, *curtain*.

	$\lvert\mathcal{E}\rvert$	avg. #targets	kitchen%	dining room %	living room%	bedroom%	bathroom%
$\mathcal{E}_{\text{small}}$	20	3.9	0.95	0.60	0.60	0.95	0.80
$\mathcal{E}_{\text{large}}$	200	3.7	1.00	0.35	0.63	0.94	0.80
$\mathcal{E}_{\text{test}}$	50	3.7	1.00	0.48	0.58	0.94	0.70

Table 3.2: Statistics of the selected environment sets for RoomNav. `RoomType`% denotes the percentage of houses containing at least one target room of type `RoomType`.

	test succ.	kitchen%	dining room %	living room%	bedroom%	bathroom%
gated-LSTM	35.8	37.9	50.4	48.0	33.5	21.2
gated-CNN	29.7	31.6	42.5	54.3	27.6	17.4

Table 3.3: Detailed test success rates for gated-CNN model and gated-LSTM model with "Mask+Depth" as input signal across different instruction concepts.

Detailed Specifications: The location information of an agent can be represented by 4 real numbers: the 3D location (x, y, z) and the rotation degree ρ of its first person view sensor, which indicates the front direction of the agent. Note that in RoomNav, the agent is not allowed to change its height z, hence the overall degree of freedom is 3.

An action can be in the form of a triple $a = (\delta_x, \delta_y, \delta_\rho)$. After taking the action a, the agent will move to a new 3D location $(x + \delta_x, y + \delta_y, z)$ with a new rotation $\rho + \delta_\rho$. The physics in House3D will detect collisions with objects under action a and in RoomNav, the agent will remain still in case of a collision. We also restrict the velocity of the agent such that $|\delta_x|, |\delta_y| \leq 0.5$ and $|\delta_\rho| \leq 30$ to ensure a smooth movement.

Continuous Action: A continuous action a consists of two parts $a = [m, r]$ where $m = (m_1, \ldots, m_4)$ is for movement and $r = (r_1, r_2)$ is for rotation. Since the velocity of the agent should be bounded, we require m, r to be a valid probability distribution. Suppose the original location of robot is (x, y, z) and the angle of camera is ρ, then after executing a, the new 3D location will be $(x + (m_1 - m_2) * 0.5, y + (m_3 - m_4) * 0.5, z)$ and the new angle is $\rho + (r_1 - r_2) * 30$.

Discrete Action: We define 12 different action triples in the form of $a_i = (\delta_x, \delta_y, \delta_\rho)$ satisfying the velocity constraints. There are 8 actions for movement: left, forward, right with two scales and two diagonal directions; and 4 actions for rotation: clockwise and counter-clockwise with two scales. In the discrete action setting, we do not allow the agent to move and rotate simultaneously.

Reward Details: In addition to the reward shaping of difference of shortest distances, we have the following rewards. When hitting an obstacle, the agent receives a penalty of 0.3. In the case of success, the winning reward is +10. In order to encourage exploration (or to prevent eternal rotation), we add a time penalty of 0.1 to the agent for each time step outside the target room. Note that since we restrict the velocity of the agent, the difference of shortest path after an action will be no more than $0.5 \times \sqrt{2} \approx 0.7$.

Additional Experiment Details

Network architectures

We apply a batch normalization layer after each layer in the CNN module. The activation function used is ReLU. The embedding dimension of concept instruction is 25.

Gated-CNN: In the CNN part, we have 4 convolution layers of 64, 64, 128, 128 channels perspective and with kernel size 5 and stride 2, as well as a fully-connected layer of 512 units. We use a linear layer to transform the concept embedding to a 512-dimension vector for gated fusion. The MLP for policy has two hidden layers of 128 and 64 units, and the MLP for Q-function has a single hidden layer of 64 units.

Gated-LSTM: In the CNN module, we have 4 convolution layers of 64, 64, 128, 128 channels each and with kernel size 5 and stride 2, as well as a fully-connected layer of 256 units. We use a linear layer to convert the concept embedding to a 256-dimension vector. The LSTM module has 256 hidden dimensions. The MLP module for policy contains two layers of 128 and 64 hidden units, and the MLP for value function has two hidden layers of 64 and 32 units.

Training parameters

We normalize each channel of the input frame to $[0, 1]$ before feeding it into the neural network. Each of the training procedures includes a weight decay of 10^{-5} and a discounted factor $\gamma = 0.95$.

DDPG: We stack $k = 5$ recent frames and use learning rate 10^4 with batch size 128. We choose $\alpha_{\text{DDPG}} = 100$ for all the settings except for the case with input signal of "RGB+Depth" on $\mathcal{E}_{\text{large}}$, where we choose $\alpha_{\text{DDPG}} = 10$. We use an entropy bonus term with coefficient 0.001 on $\mathcal{E}_{\text{small}}$ and 0.01 on $\mathcal{E}_{\text{large}}$. We use exponential average to update the target network with rate 0.001. A training update is performed every 10 time steps. The replay buffer size is 7×10^5. We run training for 80000 episodes in all. We use a linear exploration strategy in the first 30000 episodes.

A3C: We clip the reward to the range $[-1, 1]$ and use a learning rate $1e-3$ with batch size 64. We launch 120 processes on $\mathcal{E}_{\text{small}}$ and 200 on $\mathcal{E}_{\text{large}}$. During training we estimate the discounted accumulative rewards and back-propagate through time for every 30 time steps unrolled. We perform a gradient clipping of 1.0 and decay the learning rate by a factor of 1.5 when the difference of KL-divergence becomes larger than 0.01. For training on $\mathcal{E}_{\text{small}}$, we use a entropy bonus term with coefficient 0.1; while on $\mathcal{E}_{\text{large}}$, the coefficient is 0.05. α_{A3C} is 1.0. We perform 10^5 training updates and keep the best model with the highest training success rate.

Generalization over different concepts

We illustrate in Table 3.3 the detailed test success rates of our models trained on $\mathcal{E}_{\text{train}}$ with respect to each of the 5 concepts. Note that both models have similar behaviour across

concepts. In particular, "dining room" and "living room" are the easiest while "bathroom" is the hardest. We suspect that this is because dining room and living room are often with large room space and have the best connectivity to other places. By contrast, bathroom is often very small and harder to find in big houses.

Lastly, we also experiment with adding auxiliary tasks of predicting the current room type during training. We found this does not help the training performance nor the test performance. We believe it is because our reward shaping has already provided strong supervision signals.

Average steps towards success

We also measure the number of steps required for an agent in RoomNav. For all the successful episodes, we evaluate the averaged number of steps towards the final target. The numbers are shown in Table 3.4. A random agent can only succeed when it's initially spawned very close to the target, and therefore have very small number of steps towards target. Our trained agents, on the other hand, can explore in the environment and reach the target after resonable number of steps. Generally, our DDPG models takes fewer steps than our A3C models thanks to their continuous action space. But in all the settings, the number of steps required for a success is still far less than 100, namely the horizon length.

	random	concat-LSTM	gated-LSTM	concat-CNN	gated-CNN
Avg. #steps towards targets on $\mathcal{E}_{\text{small}}$ with different input signals					
RGB+Depth (train)	14.2	35.9	41.0	31.7	33.8
RGB+Depth (test)	13.3	27.1	29.8	26.1	25.3
Mask+Depth (train)	14.2	38.4	40.9	34.9	36.6
Mask+Depth (test)	13.3	31.9	34.3	26.2	30.4
Avg. #steps towards targets on $\mathcal{E}_{\text{large}}$ with different input signals					
RGB+Depth (train)	16.0	36.4	35.6	31.0	32.4
RGB+Depth (test)	13.3	34.0	33.8	24.4	25.7
Mask+Depth (train)	16.0	40.1	38.8	34.6	36.2
Mask+Depth (test)	13.3	34.8	34.3	30.6	30.9

Table 3.4: Averaged number of steps towards the target in all success trials for all the evaluated models with various input signals and different environments.

3.7 Summary

In this chapter, we introduce a new environment, House3D, which contains 45K houses with a diverse set of objects and natural layouts resembling the real-world.

In House3D, we teach an agent to accomplish semantic goals. We define RoomNav, in which an agent needs to understand a given semantic concept, interpret the comprehensive visual

signal, navigate to the target, and most importantly, succeed in a new unseen environment. We note that generalization to unseen environments was rarely studied in previous works.

To this end, we quantify the effect of various levels of augmentations, all facilitated by House3D by the means of domain randomization, multi-target training and the diversity of the environment. We resort to well established RL techniques equipped with gating to encode the task at hand. The final performance on unseen environments is much higher than baseline methods by over 8%. We hope House3D as well as our training techniques can benefit the whole RL community for building generalizable agents.

Chapter 4

Bayesian Relational Memory for Semantic Visual Navigation

In this chapter, we introduce a new memory architecture, *Bayesian Relational Memory* (BRM), to improve the generalization ability for semantic visual navigation agents in unseen environments, where an agent is given a semantic target to navigate towards. BRM takes the form of a probabilistic relation graph over semantic entities (e.g., room types), which allows (1) capturing the layout prior from training environments, i.e., *prior knowledge*, (2) estimating posterior layout at test time, i.e., *memory update*, and (3) efficient *planning* for navigation, altogether. We develop a BRM agent consisting of a BRM module for producing sub-goals and a goal-conditioned locomotion module for control. When testing in unseen environments, the BRM agent outperforms baselines that do not explicitly utilize the probabilistic relational memory structure.

4.1 Motivation

Memory is a crucial component for intelligent agents to gain extensive reasoning abilities over a long horizon. One such challenge is visual navigation, where an agent is placed to an environment with unknown layouts and room connectivity, acts on visual signal perceived from its surrounding, explores and reaches a goal position efficiently.

Due to partial observability, the agent must memorize its past experiences and react accordingly. Hence, deep learning (DL) models for visual navigation often encodes memory structures in its design. LSTM is initially used to as general-purpose implicit memory [Mirowski et al., 2017, Mirowski et al., 2018, Das et al., 2018a]. Recently, to improve the performance, explicit and navigation-specific structural memory are used [Parisotto and Salakhutdinov, 2018, Gupta et al., 2017b, Savinov et al., 2018, Gupta et al., 2017c].

Two categories exist for navigation-specific memory: the spatial memory [Parisotto and Salakhutdinov, 2018, Gupta et al., 2017b] and topological memory [Gupta et al., 2017c, Savinov et al., 2018]. The core idea of spatial memory is to extend the 1-dimensional memory in

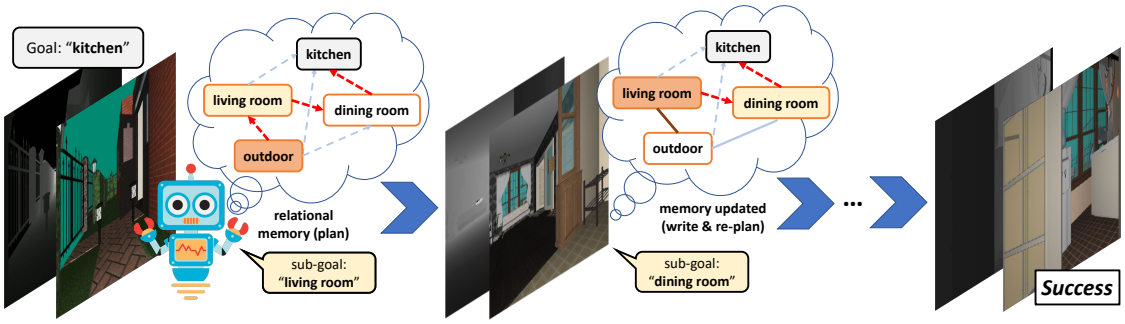


Figure 4.1: A demonstration of our task and method. The agent perceives visual signals and needs to find the kitchen, which cannot be seen from outside (leftmost). So the agent plans in its memory and conclude that it may first find a likely intermediate waypoint, i.e., living room. Then the agent repeatedly updates its belief of the environment layout, re-plans accordingly and reaches the kitchen in the end.

LSTM to a 2-dimensional matrix that represents the spatial structure of the environment, where a particular entry in the matrix corresponds to a 2D location/region in the environment. Due to its regular structure, value iteration [Tamar et al., 2016] can be applied directly for effective planning over the memory matrix.

However, planning on such spatial memory can be computationally expensive for environments with large rooms. To navigate, precise localization of an agent is often not unnecessary. Extensive psychological evidences [Savinov et al., 2018] also show that animals do not rely strongly on metric representations [Wang and Spelke, 2002, Foo et al., 2005]. Instead, humans primarily depend on a landmark-based navigation strategy, which can be supported by qualitative topological knowledge of the environment [Foo et al., 2005]. Therefore, it is reasonable to represent the memory as a topological graph where the vertices are landmarks in the environment and edges denote short-term reachability between landmarks. During navigation, a localization network is trained to identify the position of the agent and the goal w.r.t. the landmarks in the memory and an efficient graph search can be used for long-term planning.

Nevertheless, still human navigation shows superior generalization performance which cannot be explained by either spatial or topological memory. For example, first-time home visitors naturally move towards the kitchen (rather than outdoor or toilet) to get a plate; from kitchen to bedroom, they know that living room may be an intermediate waypoint. Although visually different, such *semantic knowledge*, i.e., the "close-by" relations over semantic entities, are naturally shared across environments and can be learned from previous experience to guide future navigation. In comparison, existing approaches of topological memory assume pre-exploration experiences of the environment before navigation starts [Gupta et al., 2017c, Savinov et al., 2018], provide no memory updating operations, and cannot incorporate the prior knowledge of scene layouts and configurations from previously seen environments.

In this chapter, we propose a new memory design for visual navigation, Bayesian Relational

Memory (BRM), which (1) captures the prior knowledge of scene layouts from training environments and (2) allows both efficient planning and updating during test-time exploration. BRM can be viewed as a probabilistic version of a topological memory with semantic abstractions: each node in BRM denotes a semantic concept (e.g., object category, room type, etc), which can be detected via a neural detector, and each edge denotes the relation between two concepts. In each environment, a single relation may be present or not. For each relation (edge), we can average its existences over all training environments and learn an existence probability to denote the *prior knowledge* between two semantic concepts. During exploration at test time, we can incrementally observe the existences of relations within that particular test environment. Therefore, we can use these environment specific observations to update the probability of those relations in the memory via the Bayes rule to derive the *posterior knowledge*. Additionally, we train a semantic-goal-conditioned LSTM locomotion policy for control via deep reinforcement learning (DRL), and by planning on the relation graph with posterior probabilities, the agent picks the next semantic sub-goal to navigate towards.

We evaluate our BRM method in a semantic visual navigation task on top of the House3D environment [Wu et al., 2018, Das et al., 2018a], which provides a diverse set of objects, textures and human-designed indoor scenes. The semantic scenes and entities in House3D are fully labeled (with noise), which are natural for our BRM model. In the navigation task, the agent observes first-person visual signals and needs to navigate towards a particular room type. We utilize the room types as the semantic concepts (nodes) in BRM and the "close-by" relations as the edges. We evaluate on unseen environments with random initial locations and compare our learned model against other DRL-based approaches without the BRM representation. Experimental results show that the agent equipped with BRM can achieve the semantic goal with higher success rates and fewer navigation steps.

Our contributions are as follows: (1) we proposed a new memory representation, Bayesian Relational Memory (BRM), in the form of probabilistic relation graphs over semantic concepts; (2) BRM is capable of encoding prior knowledge over training environments as well as efficient planning and updating at test time; (3) by integrating BRM into a DRL locomotion policy, we show that the generalization performances can be significantly improved.

Related Work

Navigation is one of the most fundamental problems in mobile robotics. Traditional approaches (like SLAM) build metric maps via sensory signals, which is subsequently used for planning [Elfes, 1987, Bonin-Font et al., 2008, Thrun et al., 2005, Durrant-Whyte and Bailey, 2006]. More recently, thanks to the advances of deep learning, end-to-end approaches have been applied to tackle navigation in various domains, such as mazes [Mirowski et al., 2017, Jaderberg et al., 2017], indoor scenes [Zhu et al., 2017b, Chang et al., 2017, Savva et al., 2017, Mishkin et al., 2019, Kolve et al., 2017], autonomous driving [Chen et al., 2015, Xu et al., 2017, Dosovitskiy et al., 2017], and Google street view [Mirowski et al., 2018]. There are also nice summaries of recent progresses [Anderson et al., 2018a, Mishkin et al., 2019]. We focus

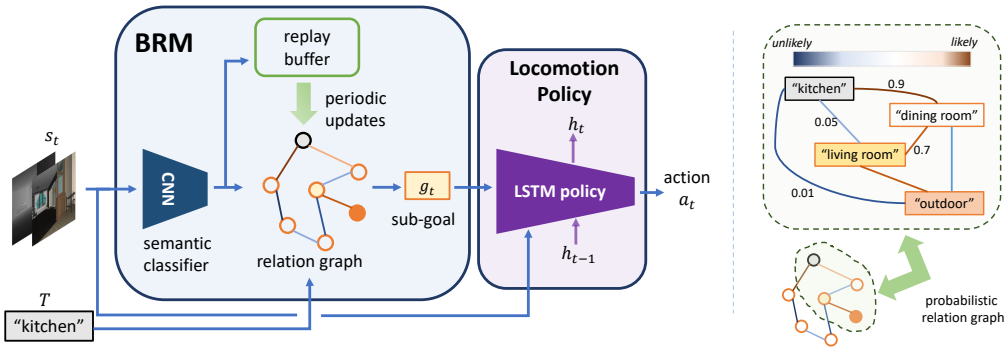


Figure 4.2: The architecture overview of our proposed navigation agent equipped with the Bayesian Relational Memory (BRM).

on indoor navigation scenario with House3D environment [Wu et al., 2018] which contains real-world-consistent relations between semantic entities and provides ground-truth labels of objects and scenes.

There are also works studying visual navigation under natural language guidance, including instruction following [Chaplot et al., 2018, Misra et al., 2017, Fried et al., 2018, Wang et al., 2018b, Anderson et al., 2018b] and question answering [Das et al., 2018b, Das et al., 2018a, Anand et al., 2018]. These tasks require the agent to understand the natural language and reason accordingly in an interactive environment. In our semantic navigation task, the goal instruction is simplified to a single semantic concept and hence we focus more on the reasoning ability of navigation agents.

In our work, the reasoning is performed on the relations over semantic concepts from visual signals. Similar ideas of using semantic knowledge to enhance reasoning have been applied to image classification [Marino et al., 2017, Wang et al., 2018c], segmentation [Torralba et al., 2003, Zhu et al., 2015], situation recognition [Li et al., 2017a], visual question answering [Wu et al., 2016a, Chen et al., 2018, Vedantam et al., 2019, Johnson et al., 2017, Hu et al., 2017], image retrieval [Johnson et al., 2015] and relation detection [Zhang et al., 2017]. Savinov et al. [Savinov et al., 2019] and Kuang et al. [Fang et al., 2019] also consider extracting visual concepts dynamically by treating every received input frame as an individual concept and storing them in the memory. The most related work to ours is a concurrent one by Wei et al. [Yang et al., 2019], which considers visual navigation towards an object category and utilizes a knowledge graph as the prior knowledge. Wei et al. [Yang et al., 2019] use a fixed graph to extract features for the target as an extra input to the locomotion without graph updating or planning. While in our work, the relation graph is used in a Bayesian manner as a representation for the memory, which unites use of prior knowledge, updating and planning altogether.

From a reinforcement learning perspective, our work is related to the model-based approaches [Doya et al., 2002, Ross and Pineau, 2008, Zhang et al., 2018, Riedmiller et al., 2018, Kurutach et al., 2018], in the sense that we model the environment layout via a relation

graph and plan on it. Our work is also related to hierarchical reinforcement learning [Sutton et al., 1999, Dayan and Hinton, 1993a, Kulkarni et al., 2016], where the controller (BRM) produces a high-level sub-goal for the sub-policy (locomotion) to pursue. Furthermore, BRM learns from multi-task training and its update operation fast adapts the prior relations to the test environment, which can be also viewed as a form of meta-learning [Finn et al., 2017a, Duan et al., 2016b, Mishra et al., 2018].

4.2 Methods

Task Setup

We consider the semantic visual navigation problem where an agent interacts with an environment with discrete time steps and navigates towards a semantic goal. In House3D, the semantic entities of interest are room types and we assume a fixed number of K categories. In the beginning of an episode, the agent is given a semantic target $T \in \mathcal{T} = \{T_1, \ldots, T_K\}$ to navigate towards for a success. At each time step t, the agent receives a visual observation s_t and produces an action $a_t \in \mathcal{A}$ conditioning on s_t and T.

We aim to learn a neural agent that can *generalize* to unseen environments. Hence, we train the agent on a training set $\mathcal{E}_{\text{train}}$, where the ground-truth labels are assumed, and validate on $\mathcal{E}_{\text{valid}}$. Evaluation is performed on another separate set of environments $\mathcal{E}_{\text{test}}$. At test time, the agent only access to the visual signal s_t without any pre-exploration experiences of the test environment.

Method Overview

The overall architecture of a BRM agent is shown in Fig. 4.2, which has two modules, the Bayesian Relational Memory (BRM) as well as an LSTM locomotor policy for control (Fig. 4.2 left). Particularly, the key component of a BRM agent is a probabilistic relation graph (Fig. 4.2 right), where each node corresponds to a particular semantic target T_i. For semantic target T_i and T_j, the edge between them denotes the "close-by" relation and the probability of that edge implies how likely T_i and T_j are close to each other in the current environment.

At a high level, the locomotion is a semantic-goal-conditioned policy which takes in both the visual input s_t and the sub-goal $g \in \mathcal{T}$ produced by the BRM module to produce actions towards g. The BRM module takes in the visual observation s_t at each time step, extracts the semantic information via a CNN detector and store them in a replay buffer. We periodically update the posterior probability of each edge in the relation graph and re-plan to produce a new sub-goal. In our work, the graph is updated every fixed number of N steps. Notably, we do not assume existences of all concepts — in case of a missing concept in particular environment, the posterior of its associated edges will approach zero as more experiences gained in an episode.

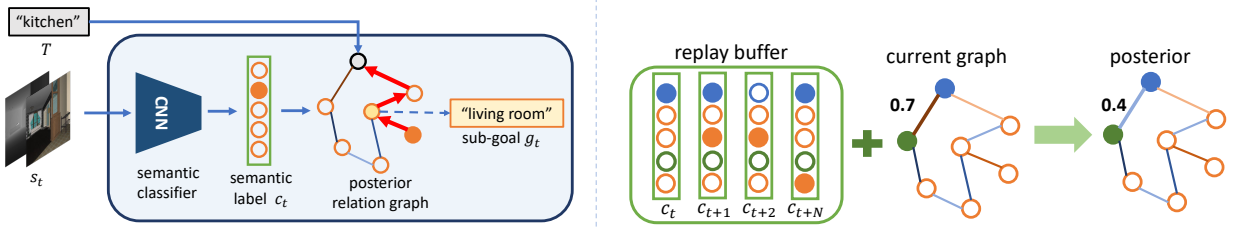


Figure 4.3: Overview of the planning (Left) and the update operation (Right) with BRM.

Bayesian Relational Memory

The BRM module consists of two parts, a semantic classifier and the most important component, a probabilistic relation graph over semantic concepts, in the form of a probabilistic graphical model allowing efficient planning and posterior updates.

Intuitively, at each time step, the agent detects the surrounding room type and maintains the probability of whether two room types T_i and T_j are "nearby" in the current environment. If the agent starts from room T_i and reaches another room T_j within a few steps, the probability of "nearby" relation between T_i and T_j should be increased[1]; otherwise the probability should be decreased. Periodically, the graph is updated and the agent finds the most likely path from the current room towards the target as a navigation guidance.

We introduce these two components in details as well as how to update and plan with BRM as follows.

Semantic classifier: The semantic classifier detects the room type label c_t for the agent's surrounding region. It can be trained by supervised learning on $\mathcal{E}_{\text{train}}$. Note that for robust room type classification, only using the first-person view image may not be enough. For example, the agent in a bedroom may face towards a wall, where the first-person image is not informative at all for the classifier, but a bed might be just behind. So we take the panoramic view as the classifier input, which consists of 4 images, $s_t^1, \ldots, s_t^4$ with different view angles. We use a 10-layer CNN with batch normalization to extract features $f(s_t^i)$ for each s_t^i and compute the attention weights over these visual features by $l_i = f(s_o^i) W_1^T W_2 \left[f(s_o^1), \ldots, f(s_o^4)\right]$ $a_i = \text{softmax}(l_i)$ with parameters W_1, W_2. Then the weighted average of these four features $\sum_i a_i f(s_t^i)$ is used for the final `sigmoid` predictions for each semantic concept T_i[2]. This results in a K-dimensional binary vector $c_t \in \{0,1\}^K$ at time t.

Probabilistic relation graph: We represent the probabilistic graph in the form of a graphical model $P(z, y; \psi)$ with latent variable z, observation variable y and parameter ψ.

Since we have K semantic concepts, there are K^3 nodes and $K(K-1)/2$ relations (edges) in the graph. Each relation is probabilistic (i.e., it may exist or not with probability)

[1] In perfect noiseless setting, the relation should with probability 1.

[2] It is a multi-label classification setting. Imagine an open kitchen with both cooking facilities and dining tables could have two labels.

[3] In fact we have $K+1$ nodes. For clarity purpose, we use K here and explain the details of the extra node later in Sec. 4.2.

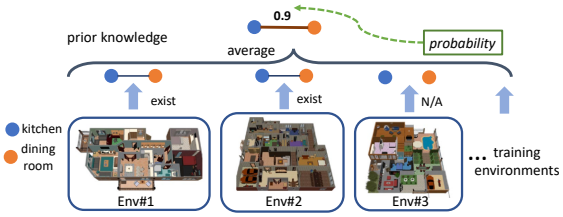


Figure 4.4: Learning the prior probability of a particular relation by analyzing the existence of that relation in each training environment and then summarizing the overall statistics.

and before entering a particular environment, we only hold a prior belief of that relation. Formally, for the relation between T_i and T_j, we adopt a Bernoulli variable $z_{i,j}$ defined by $z_{i,j} \sim \text{Bernoulli}(\psi^{\text{prior}}_{i,j})$, where parameter $\psi^{\text{prior}}_{i,j}$ denotes the prior belief of $z_{i,j}$ existing. During exploration, the agent can noisily observe $z_{i,j}$ and use the noisy observations to estimate the true value of $z_{i,j}$. We define the noisy observation model $P(y_{i,j}|z_{i,j})$ by

$$y_{i,j} \sim \begin{cases} \text{Bernoulli}(\psi^{\text{obs}}_{i,j,0}) & \text{if } z_{i,j} = 0 \\ \text{Bernoulli}(1 - \psi^{\text{obs}}_{i,j,1}) & \text{if } z_{i,j} = 1 \end{cases}, \tag{4.1}$$

where ψ^{obs} is another parameter to learn. At each time step, the agent holds an overall posterior belief $P(z|\mathcal{Y})$ of relation existences within the current environment, based on its experiences $\mathcal{Y}$, namely the samples of variable y.

Posterior update and planning: A visualization of the procedures is shown in Fig. 4.3. We assume the agent explores the current environment for a *short* horizon of N steps and stores the recent semantic signals $c_t, \ldots, c_{t+N}$ in the replay buffer. Then we compute the bit-OR operation over these binary vectors $B = c_t$ `OR` $\ldots$ `OR` c_{t+N}. B represents all the visited regions within a short (N-step) exploration period. When two targets appear concurrently in a short trajectory, they are assumed to be "close-by". For T_i and T_j with $B(T_i) = B(T_j) = 1$, T_i and T_j should be nearby in the current environment, namely a sample of $y_{i,j} = 1$; otherwise for $B(T_i) \neq B(T_j)$, we get a sample of $y_{i,j} = 0$. With all the history samples of y as $\mathcal{Y}$, we can perform posterior inference, i.e., compute posterior Bernoulli distribution $P(z_{i,j}|\mathcal{Y}_{i,j})$, for each $z_{i,j}$ by the Bayes rule.

Let $\hat{z}_{i,j} = P(z_{i,j}|\mathcal{Y}_{i,j})$ denote the posterior probability of relation over T_i and T_j. Given the current beliefs $\hat{z}$, the semantic signal c_t and the target T, we search for an optimal plan $\tau^* = \{\tau_0, \tau_1, \ldots, \tau_{m-1}, \tau_m\}$ over the graph, where $\tau_i \in \{1 \ldots K\}$ denotes an index of concepts, so that the joint belief along the path from some current position to the goal is maximized:

$$\tau^\star = \arg\max_{\tau} c_t(T_{\tau_0}) \prod_{i=1}^{m} \hat{z}_{\tau_{i-1},\tau_i}. \tag{4.2}$$

After obtaining $\tau^\star$, we execute the locomotion policy for sub-goal $g_t = T_{\tau^\star_1}$, and then periodically update the graph, clear the replay buffer and re-plan every N steps.

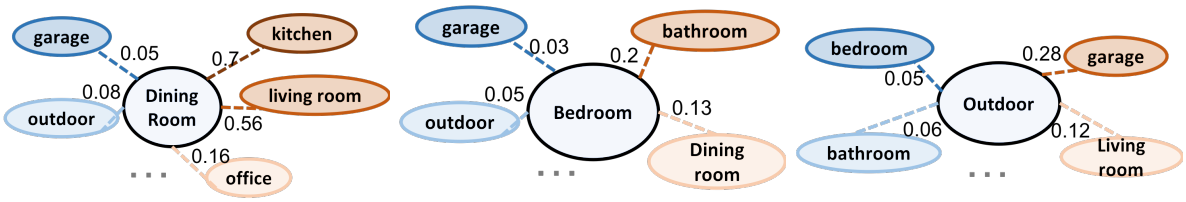


Figure 4.5: The learned prior of the relations, including the most (red) and least (blue) likely nearby rooms for dining room (Left), bedroom (Mid.) and outdoor (Right), with numbers denoting ψ^{prior}, i.e., the prior probability of two room types are nearby.

Learning the probabilistic graph: The parameter ψ has two parts: ψ^{prior} for the prior of z and ψ^{obs} for the noisy observation y.

For the prior parameter ψ^{prior}, we learn from $\mathcal{E}_{\text{train}}$ with the ground truth labels. A visualization is shown in Fig. 4.4. For each pair of room types T_i and T_j, we enumerate all training environments and run *random explorations* from some location of room type T_i. If eventually the agent reaches somewhere of room type T_j, we consider T_i and T_j are nearby and therefore a *positive* sample $z_{i,j} = 1$; otherwise a *negative* sample $z_{i,j} = 0$. Suppose $\mathcal{Z}$ denotes all the samples we obtained for z. We run maximum likelihood estimate for ψ^{prior} by maximizing $L_{\text{MLE}}(\psi^{\text{prior}}) = P(\mathcal{Z}|\psi^{\text{prior}})$. We can choose any exploration strategy such that for any "close-by" targets, they appear together in a short exploration trajectory more frequently. Random exploration is the most lightweight option among all.

The noisy observation parameter ψ^{obs} is related to the performance of the locomotion policy $\mu(\theta)$: if $\mu(\theta)$ has a higher navigation success rate, ψ^{obs} should be smaller (i.e., low noise level); when $\mu(\theta)$ is poor, ψ^{obs} should be larger (cf. Eq. (4.1)). Unfortunately, there is no direct learning supervision for ψ^{obs}. However, we can evaluate the "goodness" of a particular value of ψ^{obs} by evaluating the success rate of the overall BRM agent on $\mathcal{E}_{\text{valid}}$. Therefore, we can simply run grid search to derive the best parameter.

The Goal Conditioned Policy

We learn an LSTM locomotion policy $\mu(s_t, g; \theta)$ parameterized by θ, which conditions on observation s_t and navigates towards goal g. Following Wu et al. [Wu et al., 2018], we learn $\mu(s_t, g; \theta)$ by formulating the task as a reinforcement learning problem with shaped reward: when the agent moves towards target room g, it receives a positive reward proportional to the distance decrements; if the agent moves apart or hits an obstacle, a penalty is presented. A success reward of 10 and a time penalty of 0.1 are also assigned. We optimize the policy on $\mathcal{E}_{\text{train}}$ via the actor-critic method [Mnih et al., 2016] with a curriculum learning paradigm by periodically increasing the maximum spawn distance to the target g. Additionally, thanks to a limited set of K targets, we adopt a behavior approach [Chen et al., 2019] for improved performances: we train a separate policy $\mu_i(s_t; \theta_i)$ for each semantic target T_i and when given the sub-goal g from the BRM module, we directly execute its corresponding behavior network.

We empirically observe it performs better than the original gated-attention policy in Wu et al. [Wu et al., 2018]. Such an architecture is also common technique in other domains such as RL [Oh et al., 2015, Finn et al., 2016] and robotics [Finn and Levine, 2017].

Implementation Details

We introduce those crucial details below and defer the remaining to Section 4.4.

Nodes in the relation graph: In the previous content, we assume K pre-selected semantic concepts as graph nodes. However, it is not rare to reach some situation that cannot be categorized into any existing semantic category. In practice, we treat $c_t = \mathbf{0}$ as a special "*unknown*" concept. Hence, the BRM module actually contains $K + 1$ nodes. This is conceptually similar to natural language processing: a semantic concept is a word; the set of all concepts can be viewed as the dictionary; and $c_t = \mathbf{0}$ corresponds to the special "out-of-vocabulary" token.

Smooth temporal classification: Although the semantic classifier achieves high accuracy on validation data, the predictions may not be temporally consistent, which brings extra noise to the BRM module. For temporally smooth prediction at test time, we set a restricted threshold over the sigmoid output and apply a filtering process on top of that: the actual prediction label for room type T_i will be 1 only if the sigmoid output remains at least 0.9 confidence score for consecutively 3 time steps.

Graph learning: For learning ψ^{prior}, we run a random exploration of 300 steps and collect 50 samples for each $z_{i,j}$ per training environment. Also, learning ψ^{prior} does not depend on the locomotion $\mu(s_t, g; \theta)$ and can be reused even with different control policies. ψ^{obs} depends on the locomotion so it must be learned after $\mu(s_t, g; \theta^\star)$ is obtained.

4.3 Experiments

We experiment on the House3D environment and proceed to answer the following two questions: **(1)** Does the BRM agent captures the underlying semantic relations and behave as we expected? **(2)** Does the BRM agent generalize better in test environments than the baseline methods?

We first introduce evaluation preliminaries and baseline methods in Sec. 4.3, 4.3. Then we answer the first question qualitatively in Sec. 4.3. In Sec. 4.3, 4.3, we quantitatively show that our BRM agents generally achieve higher test success rates (i.e., better generalization) and spend fewer steps to reach the targets (i.e., more efficient) than all the baselines. We choose a fixed $N = 10$ for all the BRM agents. Ablation study on the design choices of BRM is presented in Sec. 4.3. More details can be found in Section 4.4.

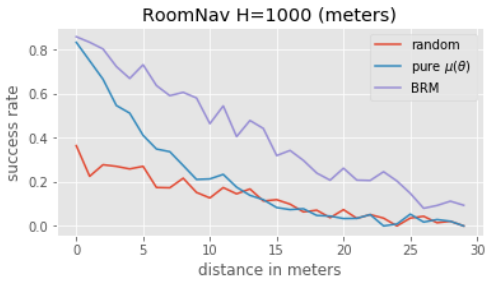


Figure 4.6: Qualitative comparison between BRM (purple), basic DRL policy (pure $\mu(\theta)$, blue) and random policy (red) with horizon $H = 1000$ (Sec. 4.3). Y-axis is the test success rate and x-axis is the distance in meters to the target. When targets become farther way from the starting point, the success rate of BRM stays high while the basic DRL policy quickly degenerates to random.

Preliminaries

We consider the RoomNav task on the House3D environment [Wu et al., 2018] where $K = 8$ room types are selected as the semantic goals, including "kitchen", "living room", "dining room", "bedroom", "bathroom", "office", "garage" and "outdoor". The House3D environment provides a success check for whether the agent has reached a specific room target or not while we also experimented on the setting of the agent predicting termination on its own (Sec. 4.3). House3D provides a training set of 200 houses, a test set of 50 houses and a validation set of 20 houses. At training time, all the approaches adopt the ground-truth semantic labels regardless of the semantic classifier.

We evaluate the generalization performances of different approaches with two metrics, *success rate* and *Success weighted by Path Length (SPL)*, under different horizons. SPL, proposed by Anderson et al. [Anderson et al., 2018a], is a function of both success rate and episode length defined by $\frac{1}{C}\sum_i S_i \frac{L_i}{\max(L_i, P_i)}$, where C is total episodes evaluated, S_i indicates whether the episode is success or not, L_i is the ground truth shortest path distance in the episode, P_i is the number of steps the agent actually took. SPL is upper-bounded by success rate and assigns more credits to agents accomplishing their tasks faster.

Baseline Methods

Random policy: The agent samples a random action per step, denoted by "random".
Pure DRL agent: This LSTM agent does not have the BRM module and directly executes the policy $\mu(s_t, T; \theta)$ throughout the entire episode, denoted by "pure $\mu(\theta)$". This is in fact the pure locomotion module. As discussed in Sec. 4.2, this is also an improved version of the original policy proposed by Wu et al. [Wu et al., 2018].
Semantic augmented agent: Comparing to the pure DRL agent, our BRM agents utilizes an extra semantic signals c_t provided by the semantic classifier in addition to the visual input

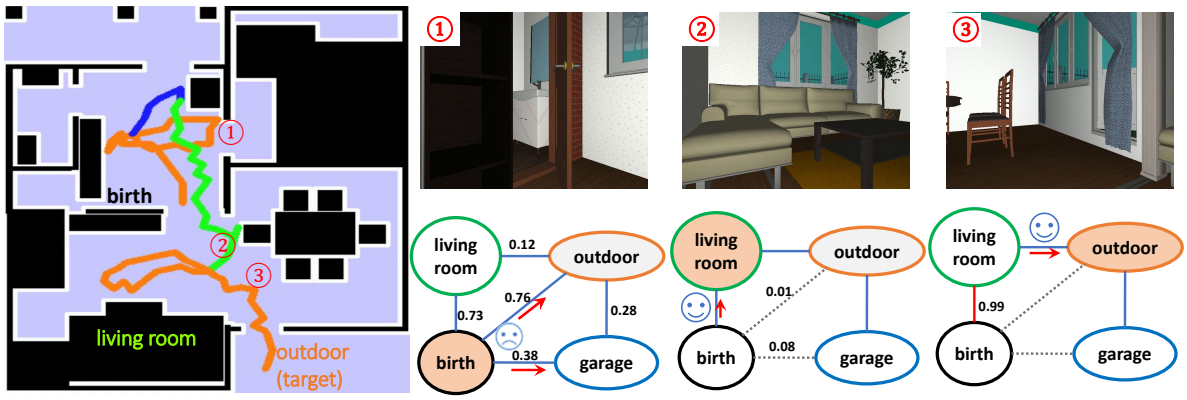


Figure 4.7: Example of a successful trajectory. The agent is spawned inside the house, targeting "outdoor". **Left**: the 2D top-down map with goal-conditioned trajectories ("outdoor" – orange; "garage" – blue; "living room" – green); **Right, 1st row**: RGB visual image; **Right, 2nd row**: the posterior of the semantic graph and the proposed sub-goals (red arrow). Initially, the agent starts by executing the locomotion for "outdoor" and then "garage" according to the prior knowledge (**1st graph**), but both fail (top orange and blue trajectories in the map). After updating its belief that garage and outdoor are not nearby (grey edges in the **2nd graph**), it then executes locomotion for "living room" with success (red arrow in the **2nd graph**, green trajectory). Finally, it executes sub-policy for"outdoor" again, explores the living room and reaches the goal (**3rd graph**, bottom orange trajectory).

s_t. Hence, we consider a semantic-aware locomotion baseline $\mu_S(\theta_s)$, which is another LSTM DRL agent that takes both s_t and c_t as input (denoted by "aug.$\mu_S(\theta_s)$").

HRL agent with an RNN controller: From a DRL perspective, our BRM agent is a hierarchical reinforcement learning (HRL) agent with the BRM module as a high-level controller producing sub-goals and the locomotion module as a low-level policy for control. Note that update and planning on BRM only depend on (1) the current semantic signal c_t, (2) the target T, and (3) the accumulative bit-`OR` feature B (see Sec. 4.2). Hence, we adopt the same locomotion $\mu(s_t, g; \theta)$ used by our BRM agent, and train an LSTM controller with 50 hidden units on $\mathcal{E}_{\text{train}}$ that takes all the necessary semantic information and produces a sub-target every N steps as well. The only difference between our BRM agent and this HRL agent is the *representation* of the controller (memory) module. The LSTM controller has access to exactly the same semantic information as BRM and uses a much more complex and generic neural model instead of a relation graph. Thus, we expect it to be a strong baseline and perform competitively to our BRM agent.

Qualitative Analysis and Case Study

In this section, we qualitatively illustrate that our BRM agent is able to learn reasonable semantic relations and behave in an interpretable manner.

Prior knowledge: Fig. 4.5 visualizes $P(z|\psi^{\text{prior}})$, the learned prior probability of relations, for 3 room types with their most and least likely nearby (connected) rooms. Darker red means more likely while darker blue implies less likely. The captured knowledge is indeed reasonable: bathroom is likely to connect to a bedroom; kitchen is often near a dining room while garage is typically outdoor.

Effectiveness of planning: The BRM agent can effectively decompose a long-term goal into a sequence of easier sub-goals via graph planning. Fig. 4.6 visualizes the test success rates of the BRM agent (BRM), random policy ("random") and the pure locomotion policy ("pure $\mu(\theta)$") for increasingly further targets over a fixed set of 5689 randomly generated test tasks. The x-axis is the shortest distance to the target in meters[4] As expected, when the target becomes more distant, all methods have lower success rates. However, as opposed to the pure locomotion policy, which quickly degenerates to random as the distance increases, the BRM agent remains a much higher success rate in general.

Case study: Fig. 4.7 shows a successful trajectory by the BRM agent, where the final target is to get out of the house. We visualize the progression of the episode, describe the plans and show the updated graph during exploration. Note that the final goal is invisible to the agent initially (frame ①) but the agent is able to plan, effectively explore the house (e.g., without ever entering the bottom-right dining room region), and eventually reach the target.

Quantitative Generalization Performances

We evaluate the generalization performances of different approaches on $\mathcal{E}_{\text{test}}$ with horizons $H = 300$ and $H = 1000$. We set $N = 10$, i.e., memory updated every 10 steps. Tab. 4.1 reports both success rates (%, percent) and SPL values (‰, per mile) over 5689 fixed test tasks. In addition to the overall performances, we also report the results under different *planning distances*, i.e., the shortest sequence of sub-goals on the ground-truth relation graph. For an accurate measurement, we ensure that there are at least 500 test tasks for each planning distance.

Our BRM agent has the highest average success rates as well as the best SPL values in *all* the cases. Notably, the margin in SPL is much more significant than that in pure success rate. More importantly, as the horizon increases, i.e., the larger number of planning computations H/N allowed, the overall performance margin (rightmost column) of BRM over the best remaining baseline strictly increases, thanks to the effectiveness of planning.

Ablation Study

In this section, we show the necessity of all the BRM components and the direction for future improvement.

Benefits of Learned Prior: In BRM, the prior $P(z|\psi^{\text{prior}})$ is learned from training houses. Tab. 4.2 evaluates BRM with an *uninformative prior* ("unif."), i.e. $\psi^{\text{prior}}_{i,j} = 0.5$. Generally,

[4]Typically one meter in shortest distance requires 3 to 4 actions.

opt. plan-steps	1	2	3	4	5	overall
avg. oracle steps	12.27	42.53	61.09	72.47	63.74	46.86
Horizon $H = 300$						
random	20.5 / 15.9	6.9 / 16.7	3.8 / 10.7	1.6 / 4.2	3.0 / 8.8	7.2 / 13.6
pure $\mu(\theta)$	49.4 / 47.6	11.8 / 27.6	2.0 / 4.8	2.6 / 10.8	4.2 / 13.2	13.1 / 22.9
aug.$\mu_S(\theta)$	47.8 / 45.3	11.4 / 23.1	3.0 / 7.8	3.4 / 8.1	4.4 / 11.2	13.0 / 20.5
RNN control.	55.0 / 49.8	20.0 / 40.8	8.0 / 20.1	5.2 / 15.2	11.0 / 25.2	19.9 / 34.2
BRM	**57.8 / 65.4**	**24.4 / 54.3**	**10.5 / 28.3**	**5.8 / 18.6**	**11.2 / 29.8**	**23.1 / 45.3**
Horizon $H = 1000$						
plan-dist	1	2	3	4	5	avg.
random	24.3 / 17.6	13.5 / 20.3	9.1 / 14.3	8.0 / 9.3	7.0 / 11.5	13.0 / 17.0
pure $\mu(\theta)$	60.8 / 47.6	23.3 / 27.6	7.6 / 4.8	8.2 / 10.8	11.0 / 13.2	22.5 / 22.9
aug.$\mu_S(\theta)$	61.3 / 50.1	23.0 / 26.2	9.4 / 12.0	5.8 / 9.6	9.0 / 13.6	22.4 / 23.8
RNN control.	71.0 / 58.0	39.6 / 51.3	24.1 / 32.7	16.6 / 25.6	23.2 / 39.6	37.0 / 45.2
BRM	**73.7 / 74.9**	**43.6 / 66.0**	**29.2 / 44.9**	**20.4 / 27.1**	**28.4 / 42.5**	**41.1 / 57.5**

Table 4.1: Metrics of ***Success Rate(%) / SPL(‰, per mile)*** evaluating the generalization performances of BRM and all the baseline approaches (Sec. 4.3). Here $N = 10$. "*plan-steps*" denotes the shortest planning distance in the ground truth relation graph. "oracle steps" denotes the reference shortest steps required to reach the goal. Our BRM agents have the highest success rates and the best SPL values in all the cases. More importantly, as the horizon increases, which allows more planning, BRM outperforms the baselines more.

BRM (H=300)	unif. (H=300)	BRM (H=1k)	unif. (H=1k)
23.1 / 45.3	20.9 / 39.4	**41.1 / 57.5**	40.4 / 56.6

Table 4.2: ***Success Rate(%) / SPL(‰)***: performances of BRM with learned and uninformative prior. $N = 10$, $H = 300, 1000$ ("1k").

the learned prior leads to better success rates and SPL values. Notably, when horizon becomes longer, the gap becomes much smaller, since the graph will converge to the true posterior with more memory updates.

Source of Error: Our approach has two modules, a BRM module for planning and a locomotion module for control. We study the errors caused by each of these components by introducing (1) a hand-designed (imperfect) oracle locomotion, which automatically gets closer to nearby targets, and (2) an optimal planner using House3D labels. The evaluation results are show in Tab. 4.3, where the oracle locomotion drastically boosts the results while the BRM performance is close to the optimal planner. This indicates that the error is mainly from locomotion – it is extremely challenging to learn a single neural navigator, which motivates our work to decompose a long-term task into sub-tasks.

Choice of N: The oracle steps for reaching a nearby, i.e., 1-plan-step, target is around 12.27 (top in Tab. 4.1), so we choose a slightly smaller value $N = 10$ as the re-planning step

Src. of Err.	LSTM locomotion	oracle locomotion
BRM	41.1 / 57.5	88.6 / N.A.
opt. plan	46.3 / 62.5	96.7 / N.A.

Table 4.3: ***Success Rate(%) / SPL(‰)***: performances with an optimal planner and a hand-designed locomotion. H=1000, N=10.

Choice of N	$N = 10$	$N = 30$	$N = 50$
BRM	**41.1 / 57.5**	29.7 / 35.2	27.4 / 32.2
RNN control.	37.0 / 45.2	28.2 / 27.7	26.5 / 26.7

Table 4.4: ***Success Rate(%) / SPL(‰)***: performances with different choices of N under horizon $H = 1000$.

Horizon $H = 1000$ with Terminate Action				
random	pure $\mu(\theta)$	aug.$\mu_S(\theta_s)$	RNN cont.	BRM
1.8/1.2	8.6/9.0	8.2/8.8	14.5/16.3	**17.3/23.0**

Table 4.5: ***Success Rate(%) / SPL(‰)*** with **terminate action** evaluating the generalization performances of BRM and baseline agents with horizon $H = 1000$ and $N = 10$. Our BRM agent achieves the best performances under both metrics.

size. We investigate other choices of N in Tab. 4.4. Larger N results in more significant performance drops. Notably, our BRM agent consistently outperforms RNN controller under different parameter choices.

Evaluation with Terminate Action

In the previous studies, the success of an episode is determined by the House3D environment automatically. It is suggested by Anderson et al. [Anderson et al., 2018a] that a real-world agent should be aware of its goal and determine whether to stop by itself. In this section, we evaluate the BRM agent and all previous baselines under this setting: a success will be counted only if the agent terminates the episode correctly in a target room on its own.

There are two ways to include the terminate action: (1) expand the action space with an extra stop action; (2) train a separate termination checker. We observe (2) leads to much better practical performances, which is also reported by Pathak et al. [Pathak et al., 2018]. In our experiments, we simply use the semantic classifier as our termination checker.

The results are summarized in Tab. 4.5, where we use a long horizon $H = 1000$ to allow the agents to have enough time to self-terminate. Similarly, BRM achieves the best performance in both success rate and SPL metric.

Discussions

Success rate and SPL: In Tab. 4.1, the SPL values are typically much smaller than the success rates, namely BRM uses much more steps than the reference shortest path. This is not surprising due to the strong partial observability in the RoomNav task. As a concrete example in Fig. 4.7, the optimal path from birthplace (near ①) to the outdoor (near ③) is extremely short if we know the top-down view in advance. However, the outdoor region is out of the agent's sight (frame ①) and the agent has to *explore* the nearby regions before it sees the door towards the outside (frame ③). Planning and updates on BRM helps guide the agent to explore the unknown house more effectively, which helps lead to higher SPL values. But overall the agent still suffers from the fundamental challenge of partial observability.

Pre-selected concepts: In this work, we focus on the memory representation and simply assume we know all the semantic concepts in advance. It is also possible to generalize to unseen concepts by leveraging the word embedding and knowledge graph from NLP community [Yang et al., 2019]. It is also feasible to directly discover general semantic concepts from $\mathcal{E}_{\text{train}}$ via unsupervised learning by leveraging the rich semantic information (e.g., object categories, room types, etc) within the visual input. We leave this to our future work.

4.4 Additional Details

Video Demo

A video demo visualizing a successful navigation trajectory by BRM can be found at the following url:

https://drive.google.com/file/d/1vCFQZfFK1X6WJacrQID2kMQVzRD4WeTs/view?usp=sharing.

Environment Details

In RoomNav the 8 targets are: kitchen, living room, dining room, bedroom, bathroom, office, garage and outdoor. We inherit the success measure of "see" from [Wu et al., 2018]: the agent needs to see some corresponding object for at least 450 pixels in the input frame and stay in the target area for at least 3 time steps.

Originally the House3D environment supports a set of 13 discrete actions. Here we reduce it to 9 actions: large forward, forward, left-forward, right-forward, large left rotate, large right rotate, left rotate, right rotate and stay still. More environment details can be found in Section 3.6 from the previous Chapter. We also implemented a faster and customized variant of the House3D environment, which is available at https://github.com/jxwuyi/House3D/tree/C++.

Evaluation Details

We measure the success rate on $\mathcal{E}_{\text{test}}$ over 5689 test episodes, which consists of 5000 randomly generated configurations and 689 specialized for faraway targets to increase the confidence of

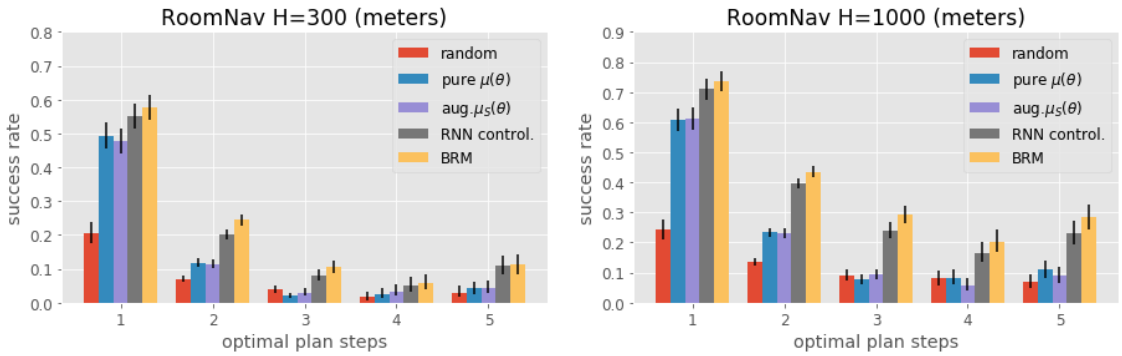


Figure 4.8: Comparing BRM with baselines in success rate with confidence interval. Approaches of interest include random policy (red), pure LSTM policy (blue), the semantic-aware LSTM policy (purple), the hierarchical policy (grey) and BRM (yellow). In all plots, the y-axis is success rate while the x-axis is the optimal planning distance. BRM outperforms all baselines and the gap becomes more significant when horizon increase, namely, more planning computations.

measured success rates. These 689 episodes are generated such that for each plan-distance, there are at least 500 evaluation episodes. Each test episode has a fixed configuration for a fair comparison between different approaches, i.e., the agent will always start from the same location with the same target in that episode. Note that we always ensure that (1) the target is connected to the birthplace of the agent, and (2) the the birthplace of the agent is never within the target room. In addition to the detailed numbers in Table 1,we visualize the success rates with *confidence intervals* for BRM and baseline methods in Figure 4.8. The confidence interval is obtained by fitting a binomial distribution.

Ablation Study: the Semantic Detector

In BRM, we use a CNN detector to extract the semantic signals at test time. Here we also evaluate the performances of all the approaches using the oracle signals from the House3D environment. The results are in Table 4.6, where we also include the BRM agent using CNN detector as a reference. Generally, using both the ground truth signal and using the CNN detector yield comparable overall performances in both metrics of success rate and SPL. They all consistently outperform all the baseline methods, which indicates that the probabilistic relational graph is robust over the noise on semantic signals (the robustness if controlled by ψ^{obs}). One interesting observation is that there are many cases, using CNN detector produces better results than using the ground truth signals. We hypothesis that this is because the semantic labels in House3D is noisy and therefore a well-trained CNN detector will not be influenced by the noisy labels at test time.

plan-dist	1	2	3	4	5	overall
Horizon $H = 300$						
plan-dist	1	2	3	4	5	avg.
random	20.5 / 15.9	6.9 / 16.7	3.8 / 10.7	1.6 / 4.2	3.0 / 8.8	7.2 / 13.6
pure $\mu(\theta)$	49.4 / 47.6	11.8 / 27.6	2.0 / 4.8	2.6 / 10.8	4.2 / 13.2	13.1 / 22.9
aug.$\mu_S(\theta)$ (true)	51.9 / **66.4**	11.1 / 24.2	3.3 / 7.8	2.4 / 6.0	3.0 / 8.7	13.2 / 23.3
RNN ctrl. (true)	54.9 / 48.1	20.2 / 37.7	8.2 / 22.5	5.6 / 13.8	9.8 / 22.7	20.0 / 32.6
BRM (true)	**58.8** / 60.7	**25.3** / **55.6**	10.4 / 26.9	**7.6** / **22.2**	9.2 / 23.4	**23.6** / 44.9
BRM (CNN)	57.8 / 65.4	24.4 / 54.3	**10.5** / **28.3**	5.8 / 18.6	**11.2** / **29.8**	23.1 / **45.3**
Horizon $H = 1000$						
plan-dist	1	2	3	4	5	avg.
random	24.3 / 17.6	13.5 / 20.3	9.1 / 14.3	8.0 / 9.3	7.0 / 11.5	13.0 / 17.0
pure $\mu(\theta)$	60.8 / 47.6	23.3 / 27.6	7.6 / 4.8	8.2 / 10.8	11.0 / 13.2	22.5 / 22.9
aug.$\mu_S(\theta)$ (true)	62.4 / 61.3	22.9 / 30.7	8.9 / 14.3	7.2 / 12.8	9.0 / 11.4	22.5 / 28.1
RNN ctrl. (true)	70.2 / 51.3	40.8 / 48.6	22.8 / 32.2	16.4 / 23.4	24.2 / 41.0	37.4 / 42.9
BRM (true)	70.3 / 61.8	**44.9** / **70.5**	**31.7** / **50.8**	19.0 / **33.3**	28.0 / 42.2	**41.7** / **59.8**
BRM (CNN)	**73.7** / **74.9**	43.6 / 66.0	29.2 / 44.9	**20.4** / 27.1	**28.4** / **42.5**	41.1 / 57.5

Table 4.6: ***Success Rate(%) / SPL(‰)***: we evaluate the performances of BRM and baselines agents using the ground truth oracle semantic signals provided by the environments. We also include the performance of the original BRM agent using CNN detector as a reference. The performance of BRM-CNN agents is comparable to BRM-true agents and sometimes even better. More discussions are in Sec. 4.4.

Additional Results on Episode Length

We illustrate the ground truth shortest distance information as well as the average episode length of success episodes for all the approaches. The results are shown in Table 4.7. The average ground truth shortest path is around 46.86 steps. Note that the agent has 9 actions per step and suffers from strong partial observability, which indicates the difficulty of the task.

Additional Implementation Details

The source code is available at https://github.com/jxwuyi/HouseNavAgent.

Learning the LSTM Locomotion

Policy Architecture: We utilize the same policy architecture and settings as [Wu et al., 2018]: we have 4 convolution layers of 64, 64, 128, 128 channels each and with kernel size 5 and stride 2, an MLP layer of 256 units, an LSTM cell of 256 units, two MLP layers of 126 and 64 units for policy head and another 2 MLP layers of 64 and 32 units for value head. Batch normalization is applied to all the layers before LSTM. Activation is ReLU. The a only

Average Ground Truth Shortest Path Length						
plan-dist	1	2	3	4	5	overall
Oracle	12.27	42.53	61.09	72.47	63.74	46.86
Average Successful Episode Length						
plan-dist	1	2	3	4	5	overall
Horizon $H = 300$						
random	34.0	112.7	143.8	148.0	149.7	89.8
pure $\mu(\theta)$	55.2	107.0	127.9	140.8	139.4	84.7
aug.$\mu_S(\theta)$	49.7	112.5	159.9	179.1	176.8	89.2
RNN control.	65.0	132.3	157.2	142.7	144.1	111.8
BRM	56.5	124.4	167.8	150.7	127.6	107.7
Horizon $H = 1000$						
random	121.7	354.7	426.6	532.8	409.5	322.1
pure $\mu(\theta)$	55.2	107.0	127.9	140.8	139.4	84.7
aug.$\mu_S(\theta)$	163.1	360.9	471.9	460.7	432.5	307.1
RNN control.	174.0	368.4	465.3	466.6	397.6	339.5
BRM	172.9	350.5	460.0	512.3	418.1	337.0

Table 4.7: Average successful episode length for different approaches. The length of shortest path reflects the strong difficulty of this task.

difference is that the original policy uses a gated attention mechanism for target conditioning while we use a behavior approach by training a separate sub-policy for each semantic target.

For the semantic augmented policy, we feed the semantic information to the MLP layer before LSTM.

Hyperparameters: We run a parallel version of A2C [Mnih et al., 2016] with 1 optimizer and 200 parallel rollout workers, each of which simulates a particular training house. We collect a training batch of 64 trajectories with 30 continuous time steps in each iteration. We set $\gamma = 0.97$, batch size 64, learning rate 0.001 with Adam, weight decay 10^{-5} and entropy bonus 0.1. We also add the squared l_2 norm of policy logits to the total loss with a coefficient of 0.01. We normalize the advantage to mean 0 and standard deviation 1. We totally run 60000 training iterations and use the final model as our learned policy.

Reward shaping: The reward at each time step is computed by the difference of shortest paths in meters from the agent's location to the goal after taking a action. We also add a time penalty of 0.1 and a collision penalty of 0.3. When the agent reaches the goal, the success reward is 10.

Curriculum learning: We run a curriculum learning by increasing the maximum of distance between agent's birth meters and target by 3 meters every 10000 iterations. We totally run 60000 training iterations and use the final model as our learned policy $\mu(\theta)$.

Building the Relational Graph

We run random exploration for 300 steps to collect a sample of z. For a particular environment, we collect totally 50 samples for each $z_{i,j}$. For all $i \neq j$, we set $\psi^{\text{obs}}_{i,j,0} = 0.001$ and $\psi^{\text{obs}}_{i,j,1} = 0.15$.

Training the CNN Semantic Extractor

We take the panoramic view as input, which consists of 4 images, $s^1_o, \ldots, s^4_o$ with different first person view angles. The only exception is that for target "outdoor", we notice that instead of using a panoramic view, simply keeping the recent 4 frames in the trajectory leads to the best prediction accuracy. We use an CNN feature extractor to extract features $f(s^i_o)$ by applying CNN layers with kernel size 3, strides $[1, 1, 1, 2, 1, 2, 1, 2, 1, 2]$ and channels $[4, 8, 16, 16, 32, 32, 64, 64, 128, 256]$. We also use relu activation and batch norm. Then we compute the attention weights over these 4 visual features by $l_i = f(s^i_o)W_1^T W_2 [f(s^1_o), \ldots, f(s^4_o)]$ and $a_i = \text{softmax}(l_i)$. Then we compute the weighted average of these four frames $g = \sum_i a_i f(s^i_o)$ and feed it to a single layer perceptron with 32 hidden units. For each semantic signal, we generate 15k positive and 15k negative training data from $\mathcal{E}_{\text{train}}$ and use Adam optimizer with learning rate $5e$-4, weight decay $1e$-5, batch size 256 and gradient clip of 5. We keep the model that has the best prediction accuracy on $\mathcal{E}_{\text{valid}}$.

For a smooth prediction during testing, we also have a hard threshold and filtering process on the CNN outputs: $s_s(T_i)$ will be 1 only if the output of CNN remains a confidence for T_i over 0.9 for consecutively 3 steps.

4.5 Summary

In this chapter, we proposed a novel design of memory architecture, Bayesian Relation Memory (BRM), for the semantic navigation task. BRM uses a semantic classifier to extract semantic labels from visual input and builds a probabilistic relation graph over the semantic concepts, which allows representing prior reachability knowledge via the edge priors, fast test-time adaptation via edge posteriors and efficient planning via graph search. Our BRM navigation agent uses BRM to produce a sub-goal for the locomotion policy to reach. Experiment results show that the BRM representation is effective and crucial for a visual navigation agent to generalize better in unseen environments.

At a high-level, our approach is general and can be applied to other tasks with semantic context information or state abstractions available to build a graph over, such as robotics manipulations where semantic concepts can be abstract states of robot arms and object locations, or video games where we can plan on semantic signals such as the game status or current resources. In future work, it is also worthwhile to investigate how to extract relations and concepts directly from training environments automatically.

Part III

Generalization in Complex Multi-Agent Games

Chapter 5

Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Games

In this chapter, we explore deep reinforcement learning methods for multi-agent domains. We begin by analyzing the difficulty of traditional algorithms in the multi-agent case: Q-learning is challenged by an inherent non-stationarity of the environment, while policy gradient suffers from a variance that increases as the number of agents grows. We then present an adaptation of actor-critic methods that considers action policies of other agents and is able to successfully learn policies that require complex multi-agent coordination. Additionally, we introduce a training regimen utilizing an ensemble of policies for each agent that leads to more robust multi-agent policies. We show the strength of our approach compared to existing methods in cooperative as well as competitive scenarios, where agent populations are able to discover various physical and informational coordination strategies.

5.1 Motivation

Reinforcement learning (RL) has recently been applied to solve challenging problems, from game playing [Mnih et al., 2015, Silver et al., 2016] to robotics [Levine et al., 2016]. In industrial applications, RL is emerging as a practical component in large scale systems such as data center cooling [DeepMind, 2016]. Most of the successes of RL have been in single agent domains, where modelling or predicting the behaviour of other actors in the environment is largely unnecessary.

However, there are a number of important applications that involve interaction between multiple agents, where emergent behavior and complexity arise from agents co-evolving together. For example, multi-robot control [Matignon et al., 2012a], the discovery of communication and language [Sukhbaatar et al., 2016, Foerster et al., 2016, Mordatch and Abbeel, 2018], multiplayer games [Peng et al., 2017], and the analysis of social dilemmas [Leibo et al., 2017] all operate in a multi-agent domain. Related problems, such as variants of hierarchical reinforcement learning [Dayan and Hinton, 1993b] can also be seen as a multi-agent

system, with multiple levels of hierarchy being equivalent to multiple agents. Additionally, multi-agent self-play has recently been shown to be a useful training paradigm [Silver et al., 2016,Sukhbaatar et al., 2018]. Successfully scaling RL to environments with multiple agents is crucial to building artificially intelligent systems that can productively interact with humans and each other.

Unfortunately, traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments. One issue is that each agent's policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent (in a way that is not explainable by changes in the agent's own policy). This presents learning stability challenges and prevents the straightforward use of past experience replay, which is crucial for stabilizing deep Q-learning. Policy gradient methods, on the other hand, usually exhibit very high variance when coordination of multiple agents is required. Alternatively, one can use model-based policy optimization which can learn optimal policies via back-propagation, but this requires a (differentiable) model of the world dynamics and assumptions about the interactions between agents. Applying these methods to competitive environments is also challenging from an optimization perspective, as evidenced by the notorious instability of adversarial training methods [Goodfellow et al., 2014a].

In this chapter, we propose a general-purpose multi-agent learning algorithm that: (1) leads to learned policies that only use local information (i.e. their own observations) at execution time, (2) does not assume a differentiable model of the environment dynamics or any particular structure on the communication method between agents, and (3) is applicable not only to cooperative interaction but to competitive or mixed interaction involving both physical and communicative behavior. The ability to act in mixed cooperative-competitive environments may be critical for intelligent agents; while competitive training provides a natural curriculum for learning [Sukhbaatar et al., 2018], agents must also exhibit cooperative behavior (e.g. with humans) at execution time.

We adopt the framework of centralized training with decentralized execution, allowing the policies to use extra information to ease training, so long as this information is not used at test time. It is unnatural to do this with Q-learning without making additional assumptions about the structure of the environment, as the Q function generally cannot contain different information at training and test time. Thus, we propose a simple extension of actor-critic policy gradient methods where the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information. After training is completed, only the local actors are used at execution phase, acting in a decentralized manner and equally applicable in cooperative and competitive settings.

Since the centralized critic function explicitly uses the decision-making policies of other agents, we additionally show that agents can learn approximate models of other agents online and effectively use them in their own policy learning procedure. We also introduce a method to improve the stability of multi-agent policies by training agents with an ensemble of policies, thus requiring robust interaction with a variety of collaborator and competitor policies. We empirically show the success of our approach compared to existing methods in cooperative as

well as competitive scenarios, where agent populations are able to discover complex physical and communicative coordination strategies.

Related Work

The simplest approach to learning in multi-agent settings is to use independently learning agents. This was attempted with Q-learning in [Tan, 1993], but does not perform well in practice [Matignon et al., 2012b]. As we will show, independently-learning policy gradient methods also perform poorly. One issue is that each agent's policy changes during training, resulting in a non-stationary environment and preventing the naïve application of experience replay. Previous work has attempted to address this by inputting other agent's policy parameters to the Q function [Tesauro, 2004], explicitly adding the iteration index to the replay buffer, or using importance sampling [Foerster et al., 2017]. Deep Q-learning approaches have previously been investigated in [Tampuu et al., 2017] to train competing Pong agents.

The nature of interaction between agents can either be cooperative, competitive, or both and many algorithms are designed only for a particular nature of interaction. Most studied are cooperative settings, with strategies such as optimistic and hysteretic Q function updates [Lauer and Riedmiller, 2000, Matignon et al., 2007, Omidshafiei et al., 2017], which assume that the actions of other agents are made to improve collective reward. Another approach is to indirectly arrive at cooperation via sharing of policy parameters [Gupta et al., 2017a], but this requires homogeneous agent capabilities. These algorithms are generally not applicable in competitive or mixed settings. See [Panait and Luke, 2005, Busoniu et al., 2008] for surveys of multi-agent learning approaches and applications.

Concurrently to our work, [Foerster et al., 2018b] proposed a similar idea of using policy gradient methods with a centralized critic, and test their approach on a StarCraft micromanagement task. Their approach differs from ours in the following ways: (1) they learn a single centralized critic for all agents, whereas we learn a centralized critic for each agent, allowing for agents with differing reward functions including competitive scenarios, (2) we consider environments with explicit communication between agents, (3) they combine recurrent policies with feed-forward critics, whereas our experiments use feed-forward policies (although our methods are applicable to recurrent policies), (4) we learn continuous policies whereas they learn discrete policies.

Recent work has focused on learning grounded cooperative communication protocols between agents to solve various tasks [Sukhbaatar et al., 2016, Foerster et al., 2016, Mordatch and Abbeel, 2018]. However, these methods are usually only applicable when the communication between agents is carried out over a dedicated, differentiable communication channel.

Our method requires explicitly modeling decision-making process of other agents. The importance of such modeling has been recognized by both reinforcement learning [Boutilier, 1996, Chalkiadakis and Boutilier, 2003] and cognitive science communities [Frank and Goodman, 2012]. [Hu and Wellman, 1998b] stressed the importance of being robust to the decision making process of other agents, as do others by building Bayesian models of decision making. We incorporate such robustness considerations by requiring that agents interact successfully

with an ensemble of any possible policies of other agents, improving training stability and robustness of agents after training.

Preliminary

Markov Games In this work, we consider a multi-agent extension of Markov decision processes (MDPs) called partially observable Markov games [Littman, 1994]. A Markov game for N agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent. To choose actions, each agent i uses a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$, which produces the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$.[1] Each agent i obtains rewards as a function of the state and agent's action $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$, and receives a private observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i aims to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ where γ is a discount factor and T is the time horizon.

Q-Learning and Deep Q-Networks (DQN). Q-Learning and DQN [Mnih et al., 2015] are popular methods in reinforcement learning and have been previously applied to multi-agent settings [Foerster et al., 2016, Tesauro, 2004]. Q-Learning makes use of an action-value function for policy $\boldsymbol{\pi}$ as $Q^{\boldsymbol{\pi}}(s, a) = \mathbb{E}[R|s^t = s, a^t = a]$. This Q function can be recursively rewritten as $Q^{\boldsymbol{\pi}}(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \boldsymbol{\pi}}[Q^{\boldsymbol{\pi}}(s', a')]]$. DQN learns the action-value function Q^* corresponding to the optimal policy by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^*(s, a|\theta) - y)^2], \qquad \text{where} \qquad y = r + \gamma \max_{a'} \bar{Q}^*(s', a'), \tag{5.1}$$

where $\bar{Q}$ is a target Q function whose parameters are periodically updated with the most recent θ, which helps stabilize learning. Another crucial component of stabilizing DQN is the use of an experience replay buffer $\mathcal{D}$ containing tuples (s, a, r, s').

Q-Learning can be directly applied to multi-agent settings by having each agent i learn an independently optimal function Q_i [Tan, 1993]. However, because agents are independently updating their policies as learning progresses, the environment appears non-stationary from the view of any one agent, violating Markov assumptions required for convergence of Q-learning. Another difficulty observed in [Foerster et al., 2017] is that the experience replay buffer cannot be used in such a setting since in general, $P(s'|s, a, \boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_N) \neq P(s'|s, a, \boldsymbol{\pi}'_1, ..., \boldsymbol{\pi}'_N)$ when any $\boldsymbol{\pi}_i \neq \boldsymbol{\pi}'_i$.

Policy Gradient (PG) Algorithms. Policy gradient methods are another popular choice for a variety of RL tasks. The main idea is to directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s \sim p^{\boldsymbol{\pi}}, a \sim \boldsymbol{\pi}_\theta}[R]$ by taking steps in the direction of

[1] To minimize notation we will often omit θ from the subscript of $\boldsymbol{\pi}$.

$\nabla_\theta J(\theta)$. Using the Q function defined previously, the gradient of the policy can be written as [Sutton et al., 2000]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\pi}}, a\sim\boldsymbol{\pi}_\theta}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a|s) Q^{\boldsymbol{\pi}}(s,a)], \tag{5.2}$$

where $p^{\boldsymbol{\pi}}$ is the state distribution. The policy gradient theorem has given rise to several practical algorithms, which often differ in how they estimate $Q^{\boldsymbol{\pi}}$. For example, one can simply use a sample return $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, which leads to the REINFORCE algorithm [Williams, 1992]. Alternatively, one could learn an approximation of the true action-value function $Q^{\boldsymbol{\pi}}(s,a)$ by e.g. temporal-difference learning [Sutton and Barto, 1998]; this $Q^{\boldsymbol{\pi}}(s,a)$ is called the *critic* and leads to a variety of *actor-critic* algorithms [Sutton and Barto, 1998].

Policy gradient methods are known to exhibit high variance gradient estimates. This is exacerbated in multi-agent settings; since an agent's reward usually depends on the actions of many agents, the reward conditioned only on the agent's own actions (when the actions of other agents are not considered in the agent's optimization process) exhibits much more variability, thereby increasing the variance of its gradients. Below, we show a simple setting where the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents.

Proposition 1. *Consider N agents with binary actions: $P(a_i = 1) = \theta_i$, where $R(a_1, \ldots, a_N) = \mathbf{1}_{a_1=\cdots=a_N}$. We assume an uninformed scenario, in which agents are initialized to $\theta_i = 0.5\ \forall i$. Then, if we are estimating the gradient of the cost J with policy gradient, we have:*

$$P(\langle \hat{\nabla} J, \nabla J \rangle > 0) \propto (0.5)^N$$

where $\hat{\nabla} J$ is the policy gradient estimator from a single sample, and ∇J is the true gradient.

Proof. See Section 5.4. □

The use of baselines, such as value function baselines typically used to ameliorate high variance, is problematic in multi-agent settings due to the non-stationarity issues mentioned previously.

Deterministic Policy Gradient (DPG) Algorithms. It is also possible to extend the policy gradient framework to deterministic policies $\boldsymbol{\mu}_\theta : \mathcal{S} \mapsto \mathcal{A}$ [Silver et al., 2014]. In particular, under certain conditions we can write the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}}[R(s,a)]$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_\theta \boldsymbol{\mu}_\theta(a|s) \nabla_a Q^{\boldsymbol{\mu}}(s,a)|_{a=\boldsymbol{\mu}_\theta(s)}] \tag{5.3}$$

Since this theorem relies on $\nabla_a Q^{\boldsymbol{\mu}}(s,a)$, it requires that the action space $\mathcal{A}$ (and thus the policy $\boldsymbol{\mu}$) be continuous.

Deep deterministic policy gradient (DDPG) [Lillicrap et al., 2015] is a variant of DPG where the policy $\boldsymbol{\mu}$ and critic $Q^{\boldsymbol{\mu}}$ are approximated with deep neural networks. DDPG is an

off-policy algorithm, and samples trajectories from a replay buffer of experiences that are stored throughout training. DDPG also makes use of a target network, as in DQN [Mnih et al., 2015]. *Deep deterministic policy gradient* (DDPG) [Lillicrap et al., 2015] is a variant of DPG where the policy $\boldsymbol{\mu}$ and critic $Q^{\boldsymbol{\mu}}$ are approximated with deep neural networks. DDPG is an off-policy algorithm, and samples trajectories from a replay buffer of experiences that are stored throughout training. DDPG also makes use of a target network, as in DQN [Mnih et al., 2015].

5.2 Methods

Multi-Agent Actor Critic

We have argued in the previous section that naïve policy gradient methods perform poorly in simple multi-agent settings, and this is supported in our experiments in Section 5.3. Our goal in this section is to derive an algorithm that works well in such settings. However, we would like to operate under the following constraints: (1) the learned policies can only use local information (i.e. their own observations) at execution time, (2) we do not assume a differentiable model of the environment dynamics, unlike in [Mordatch and Abbeel, 2018], and (3) we do not assume any particular structure on the communication method between agents (that is, we don't assume a differentiable communication channel). Fulfilling the above desiderata would provide a general-purpose multi-agent learning algorithm that could be applied not just to cooperative games with explicit communication channels, but competitive games and games involving only physical interactions between agents.

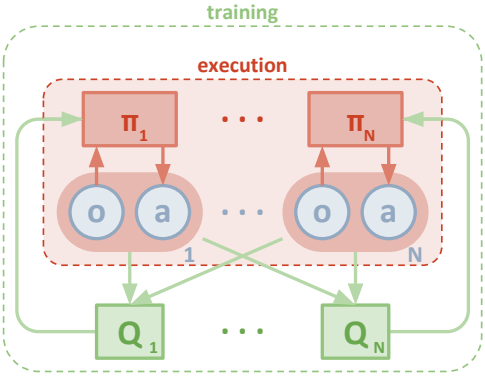


Figure 5.1: Overview of our multi-agent decentralized actor, centralized critic approach.

Similarly to [Foerster et al., 2016], we accomplish our goal by adopting the framework of centralized training with decentralized execution. Thus, we allow the policies to use extra information to ease training, so long as this information is not used at test time. It is unnatural to do this with Q-learning, as the Q function generally cannot contain different information at training and test time. Thus, we propose a simple extension of actor-critic policy gradient methods where the critic is augmented with extra information about the policies of other agents.

More concretely, consider a game with N agents with policies parameterized by $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$, and let $\boldsymbol{\pi} = \{\boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_N\}$ be the set of all agent policies. Then we can write the gradient of the expected return for agent i, $J(\theta_i) = \mathbb{E}[R_i]$ as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p^{\boldsymbol{\mu}}, a_i \sim \boldsymbol{\pi}_i}[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)]. \tag{5.4}$$

Here $Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)$ is a *centralized action-value function* that takes as input the actions of all agents, $a_1, \ldots, a_N$, in addition to some state information $\mathbf{x}$, and outputs the Q-value for agent i. In the simplest case, $\mathbf{x}$ could consist of the observations of all agents, $\mathbf{x} = (o_1, ..., o_N)$, however we could also include additional state information if available. Since each $Q_i^{\boldsymbol{\pi}}$ is learned separately, agents can have arbitrary reward structures, including conflicting rewards in a competitive setting.

We can extend the above idea to work with deterministic policies. If we now consider N continuous policies $\boldsymbol{\mu}_{\theta_i}$ w.r.t. parameters θ_i (abbreviated as $\boldsymbol{\mu}_i$), the gradient can be written as:

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}[\nabla_{\theta_i}\boldsymbol{\mu}_i(a_i|o_i)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i=\boldsymbol{\mu}_i(o_i)}], \tag{5.5}$$

Here the experience replay buffer $\mathcal{D}$ contains the tuples $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$, recording experiences of all agents. The centralized action-value function $Q_i^{\boldsymbol{\mu}}$ is updated as:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \quad y = r_i + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a'_1, \ldots, a'_N)\big|_{a'_j=\boldsymbol{\mu}'_j(o_j)}, \tag{5.6}$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_{\theta'_1}, ..., \boldsymbol{\mu}_{\theta'_N}\}$ is the set of target policies with delayed parameters θ'_i. As shown in Section 5.3, we find the centralized critic with deterministic policies works very well in practice, and refer to it as *multi-agent deep deterministic policy gradient* (MADDPG). We provide the description of the full algorithm in the Section 5.4.

A primary motivation behind MADDPG is that, if we know the actions taken by all agents, the environment is stationary even as the policies change, since

$$P(s'|s, a_1, ..., a_N, \boldsymbol{\pi}_1, ..., \boldsymbol{\pi}_N) = P(s'|s, a_1, ..., a_N) = P(s'|s, a_1, ..., a_N, \boldsymbol{\pi}'_1, ..., \boldsymbol{\pi}'_N)$$

for any $\boldsymbol{\pi}_i \neq \boldsymbol{\pi}'_i$. This is not the case if we do not explicitly condition on the actions of other agents, as done for most traditional RL methods.

Note that we require the policies of other agents to apply an update in Eq. 5.6. Knowing the observations and policies of other agents is not a particularly restrictive assumption; if our goal is to train agents to exhibit complex communicative behaviour in simulation, this information is often available to all agents. However, we can relax this assumption if necessary by learning the policies of other agents from observations — we describe a method of doing this in Section 5.2.

Inferring Policies of Other Agents

To remove the assumption of knowing other agents' policies, as required in Eq. 5.6, each agent i can additionally maintain an approximation $\hat{\boldsymbol{\mu}}_{\phi_i^j}$ (where ϕ are the parameters of the approximation; henceforth $\hat{\boldsymbol{\mu}}_i^j$) to the true policy of agent j, $\boldsymbol{\mu}_j$. This approximate policy is learned by maximizing the log probability of agent j's actions, with an entropy regularizer:

$$\mathcal{L}(\phi_i^j) = -\mathbb{E}_{o_j,a_j}\left[\log \hat{\boldsymbol{\mu}}_i^j(a_j|o_j) + \lambda H(\hat{\boldsymbol{\mu}}_i^j)\right], \tag{5.7}$$

where H is the entropy of the policy distribution. With the approximate policies, y in Eq. 5.6 can be replaced by an approximate value $\hat{y}$ calculated as follows:

$$\hat{y} = r_i + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', \hat{\boldsymbol{\mu}}_i'^1(o_1), \ldots, \boldsymbol{\mu}_i'(o_i), \ldots, \hat{\boldsymbol{\mu}}_i'^N(o_N)), \tag{5.8}$$

where $\hat{\boldsymbol{\mu}}_i'^j$ denotes the target network for the approximate policy $\hat{\boldsymbol{\mu}}_i^j$. Note that Eq. 5.7 can be optimized in a completely online fashion: before updating $Q_i^{\boldsymbol{\mu}}$, the centralized Q function, we take the latest samples of each agent j from the replay buffer to perform a single gradient step to update ϕ_i^j. Note also that, in the above equation, we input the action log probabilities of each agent directly into Q, rather than sampling.

Agents with Policy Ensembles

As previously mentioned, a recurring problem in multi-agent reinforcement learning is the environment non-stationarity due to the agents' changing policies. This is particularly true in competitive settings, where agents can derive a strong policy by overfitting to the behavior of their competitors. Such policies are undesirable as they are brittle and may fail when the competitors alter strategies.

To obtain multi-agent policies that are more robust to changes in the policy of competing agents, we propose to train a collection of K different sub-policies. At each episode, we randomly select one particular sub-policy for each agent to execute. Suppose that policy $\boldsymbol{\mu}_i$ is an ensemble of K different sub-policies with sub-policy k denoted by $\boldsymbol{\mu}_{\theta_i^{(k)}}$ (denoted as $\boldsymbol{\mu}_i^{(k)}$). For agent i, we are then maximizing the ensemble objective:

$$J_e(\boldsymbol{\mu}_i) = \mathbb{E}_{k\sim\text{unif}(1,K), s\sim p^{\boldsymbol{\mu}}, a\sim\boldsymbol{\mu}_i^{(k)}}\left[R_i(s,a)\right]. \tag{5.9}$$

Since different sub-policies will be executed in different episodes, we maintain a replay buffer $\mathcal{D}_i^{(k)}$ for each sub-policy $\boldsymbol{\mu}_i^{(k)}$ of agent i. Accordingly, we can derive the gradient of the ensemble objective with respect to $\theta_i^{(k)}$ as follows:

$$\nabla_{\theta_i^{(k)}} J_e(\boldsymbol{\mu}_i) = \frac{1}{K}\mathbb{E}_{\mathbf{x},a\sim\mathcal{D}_i^{(k)}}\left[\nabla_{\theta_i^{(k)}}\boldsymbol{\mu}_i^{(k)}(a_i|o_i)\nabla_{a_i}Q^{\boldsymbol{\mu}_i}\left(\mathbf{x}, a_1, \ldots, a_N\right)\Big|_{a_i=\boldsymbol{\mu}_i^{(k)}(o_i)}\right]. \tag{5.10}$$

5.3 Experiments

Videos of our experiments can be viewed at https://sites.google.com/site/multiagentac/.

Environments

To perform our experiments, we adopt the grounded communication environment proposed in [Mordatch and Abbeel, 2018][2], which consists of N agents and L landmarks inhabiting a

[2]Code can be found here at https://github.com/openai/multiagent-particle-envs

two-dimensional world with continuous space and discrete time. Agents may take physical actions in the environment and communication actions that get broadcasted to other agents. Unlike [Mordatch and Abbeel, 2018], we do not assume that all agents have identical action and observation spaces, or act according to the same policy $\boldsymbol{\pi}$. We also consider games that are both cooperative (all agents must maximize a shared return) and competitive (agents have conflicting goals). Some environments require explicit communication between agents in order to achieve the best reward, while in other environments agents can only perform physical actions. We provide details for each environment below.

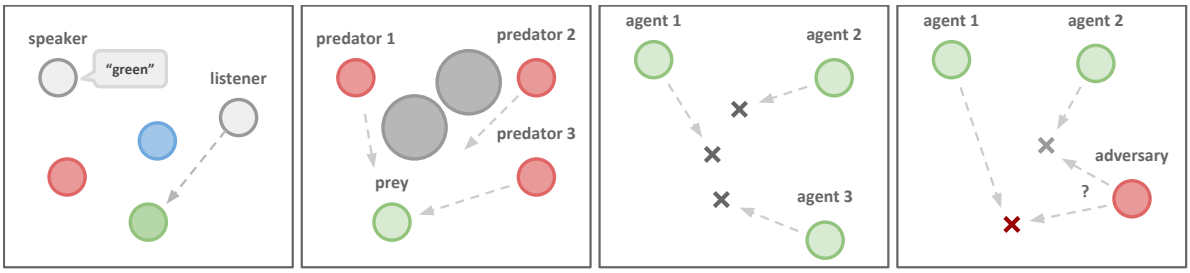


Figure 5.2: Illustrations of the experimental environment and some tasks we consider, including a) *Cooperative Communication* b) *Predator-Prey* c) *Cooperative Navigation* d) *Physical Deception*. See webpage for videos of all experimental results.

Cooperative communication. This task consists of two cooperative agents, a speaker and a listener, who are placed in an environment with three landmarks of differing colors. At each episode, the listener must navigate to a landmark of a particular color, and obtains reward based on its distance to the correct landmark. However, while the listener can observe the relative position and color of the landmarks, it does not know which landmark it must navigate to. Conversely, the speaker's observation consists of the correct landmark color, and it can produce a communication output at each time step which is observed by the listener. Thus, the speaker must learn to output the landmark colour based on the motions of the listener. Although this problem is relatively simple, as we show in Section 5.3 it poses a significant challenge to traditional RL algorithms.

Cooperative navigation. In this environment, agents must cooperate through physical actions to reach a set of L landmarks. Agents observe the relative positions of other agents and landmarks, and are collectively rewarded based on the proximity of any agent to each landmark. In other words, the agents have to 'cover' all of the landmarks. Further, the agents occupy significant physical space and are penalized when colliding with each other. Our agents learn to infer the landmark they must cover, and move there while avoiding other agents.

Keep-away. This scenario consists of L landmarks including a target landmark, N cooperating agents who know the target landmark and are rewarded based on their distance to the target, and M *adversarial* agents who must prevent the cooperating agents from reaching the target. Adversaries accomplish this by physically pushing the agents away from the landmark, temporarily occupying it. While the adversaries are also rewarded based on

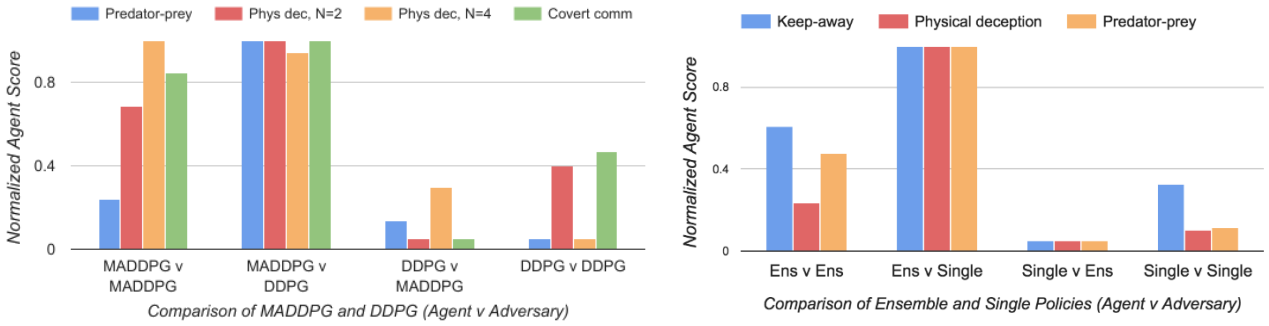


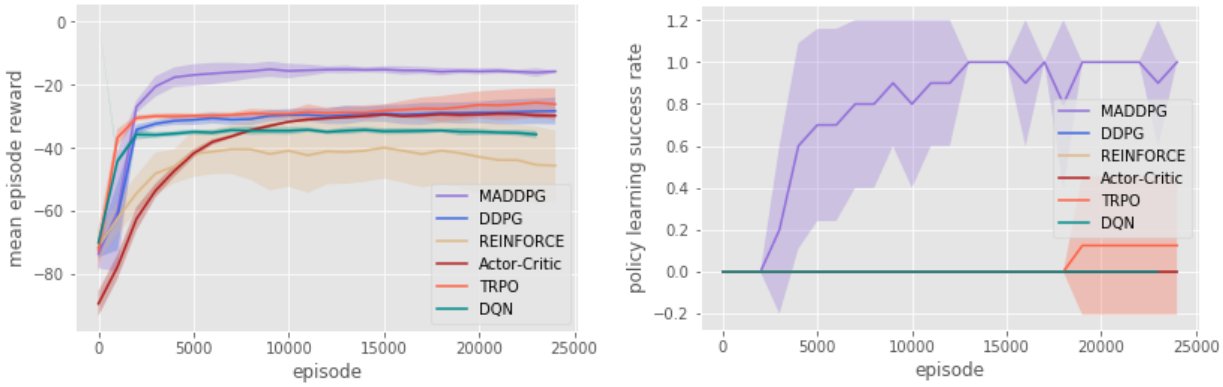
Figure 5.3: Comparison between MADDPG and DDPG (left), and between single policy MADDPG and ensemble MADDPG (right) on the competitive environments. Each bar cluster shows the 0-1 normalized score for a set of competing policies (agent v adversary), where a higher score is better for the agent. In all cases, MADDPG outperforms DDPG when directly pitted against it, and similarly for the ensemble against the single MADDPG policies. Full results are given in Section 5.4.

their distance to the target landmark, they do not know the correct target; this must be inferred from the movements of the agents.

Physical deception. Here, N agents cooperate to reach a single target landmark from a total of N landmarks. They are rewarded based on the minimum distance of any agent to the target (so only one agent needs to reach the target landmark). However, a lone adversary also desires to reach the target landmark; the catch is that the adversary does not know which of the landmarks is the correct one. Thus the cooperating agents, who are penalized based on the adversary distance to the target, learn to spread out and cover all landmarks so as to deceive the adversary.

Predator-prey. In this variant of the classic predator-prey game, N slower cooperating agents must chase the faster adversary around a randomly generated environment with L large landmarks impeding the way. Each time the cooperative agents collide with an adversary, the agents are rewarded while the adversary is penalized. Agents observe the relative positions and velocities of the agents, and the positions of the landmarks.

Covert communication. This is an adversarial communication environment, where a speaker agent ('Alice') must communicate a message to a listener agent ('Bob'), who must reconstruct the message at the other end. However, an adversarial agent ('Eve') is also observing the channel, and wants to reconstruct the message — Alice and Bob are penalized based on Eve's reconstruction, and thus Alice must encode her message using a randomly generated *key*, known only to Alice and Bob. This is similar to the cryptography environment considered in [Abadi and Andersen, 2016].



(a) Agent reward (b) Success rate

Figure 5.4: Evaluation on cooperative communication after 25000 episodes.

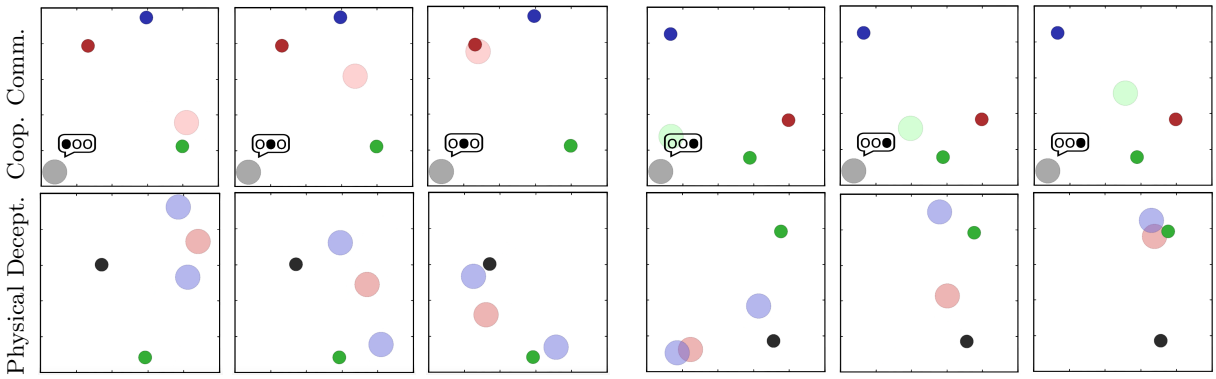
Comparison to Decentralized Reinforcement Learning Methods

We implement our MADDPG algorithm and evaluate it on the environments presented in Section 5.3. Unless otherwise specified, our policies are parameterized by a two-layer ReLU MLP with 64 units per layer. To support discrete communication messages, we use the Gumbel-Softmax estimator [Jang et al., 2017]. To evaluate the quality of policies learned in competitive settings, we pitch MADDPG agents against DDPG agents, and compare the resulting success of the agents and adversaries in the environment. We train our models until convergence, and then evaluate them by averaging various metrics for 1000 further iterations. We provide the tables and details of our results on all environments in the Section 5.4, and summarize them here.

We first examine the cooperative communication scenario. Despite the simplicity of the task (the speaker only needs to learn to output its observation), traditional RL methods such as DQN, Actor-Critic, a first-order implementation of TRPO, and DDPG all fail to learn the correct behaviour (measured by whether the listener is within a short distance from the target landmark). In practice we observed that the listener learns to ignore the speaker and simply moves to the middle of all observed landmarks. We plot the learning curves over 25000 episodes for various approaches in Figure 5.4a.

We hypothesize that a primary reason for the failure of traditional RL methods in this (and other) multi-agent settings is the lack of a consistent gradient signal. For example, if the speaker utters the correct symbol while the listener moves in the wrong direction, the speaker is penalized. This problem is exacerbated as the number of time steps grows: we observed that traditional policy gradient methods can learn when the objective of the listener is simply to reconstruct the observation of the speaker in a single time step, or if the initial positions of agents and landmarks are fixed and evenly distributed. This indicates that many of the multi-agent methods previously proposed for scenarios with short time horizons (e.g. [Lazaridou et al., 2017]) may not generalize to more complex tasks.

Conversely, MADDPG agents can learn coordinated behaviour more easily via the cen-



(a) MADDPG (b) DDPG

Figure 5.5: Comparison between MADDPG (left) and DDPG (right) on the cooperative communication (CC) and physical deception (PD) environments at $t = 0$, 5, and 25. Small dark circles indicate landmarks. In CC, the grey agent is the speaker, and the color of the listener indicates the target landmark. In PD, the blue agents are trying to deceive the red adversary, while covering the target landmark (in green). MADDPG learns the correct behavior in both cases: in CC the speaker learns to output the target landmark color to direct the listener, while in PD the agents learn to cover both landmarks to confuse the adversary. DDPG (and other RL algorithms) struggles in these settings: in CC the speaker always repeats the same utterance and the listener moves to the middle of the landmarks, and in PP one agent greedily pursues the green landmark (and is followed by the adversary) while the othe agent scatters. See video for full trajectories.

tralized critic. In the cooperative communication environment, MADDPG is able to reliably learn the correct listener and speaker policies, and the listener is often (84.0% of the time) able to navigate to the target.

A similar situation arises for the physical deception task: when the cooperating agents are trained with MADDPG, they are able to successfully deceive the adversary by covering all of the landmarks around 94% of the time when $L = 2$ (Figure 5). Furthermore, the adversary success is quite low, especially when the adversary is trained with DDPG (16.4% when $L = 2$). This contrasts sharply with the behaviour learned by the cooperating DDPG agents, who are unable to deceive MADDPG adversaries in any scenario, and do not even deceive other DDPG agents when $L = 4$.

While the cooperative navigation and predator-prey tasks have a less stark divide between success and failure, in both cases the MADDPG agents outperform the DDPG agents. In cooperative navigation, MADDPG agents have a slightly smaller average distance to each landmark, but have almost half the average number of collisions per episode (when $N = 2$) compared to DDPG agents due to the ease of coordination. Similarly, MADDPG predators are far more successful at chasing DDPG prey (16.1 collisions/episode) than the converse (10.3 collisions/episode).

In the covert communication environment, we found that Bob trained with both MADDPG

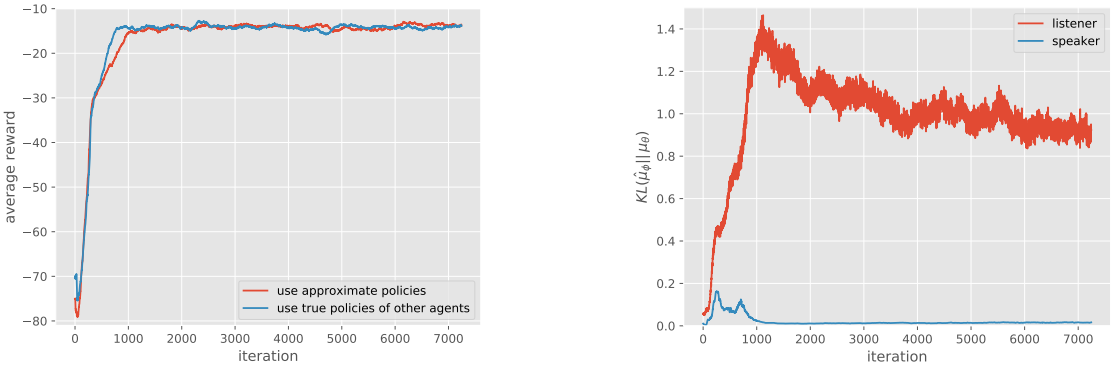


Figure 5.6: Effectiveness of learning by approximating policies of other agents in the cooperative communication scenario. *Left:* plot of the reward over number of iterations; MADDPG agents quickly learn to solve the task when approximating the policies of others. *Right:* KL divergence between the approximate policies and the true policies.

and DDPG out-performs Eve in terms of reconstructing Alice's message. However, Bob trained with MADDPG achieves a larger relative success rate compared with DDPG (52.4% to 25.1%). Further, only Alice trained with MADDPG can encode her message such that Eve achieves near-random reconstruction accuracy. The learning curve (a sample plot is shown in Section 5.4) shows that the oscillation due to the competitive nature of the environment often cannot be overcome with common decentralized RL methods. We emphasize that we do not use any of the tricks required for the cryptography environment from [Abadi and Andersen, 2016], including modifying Eve's loss function, alternating agent and adversary training, and using a hybrid 'mix & transform' feed-forward and convolutional architecture.

Effect of Learning Polices of Other Agents

We evaluate the effectiveness of learning the policies of other agents in the cooperative communication environment, following the same hyperparameters as the previous experiments and setting $\lambda = 0.001$ in Eq. 5.7. The results are shown in Figure 5.6. We observe that despite not fitting the policies of other agents perfectly (in particular, the approximate listener policy learned by the speaker has a fairly large KL divergence to the true policy), learning with approximated policies is able to achieve the same success rate as using the true policy, without a significant slowdown in convergence.

Effect of Training with Policy Ensembles

We focus on the effectiveness of policy ensembles in competitive environments, including keep-away, cooperative navigation, and predator-prey. We choose $K = 3$ sub-policies for the keep-away and cooperative navigation environments, and $K = 2$ for predator-prey. To improve convergence speed, we enforce that the cooperative agents should have the same

policies at each episode, and similarly for the adversaries. To evaluate the approach, we measure the performance of ensemble policies and single policies in the roles of both agent and adversary. The results are shown on the right side of Figure 5.3. We observe that agents with policy ensembles are stronger than those with a single policy. In particular, when pitting ensemble agents against single policy adversaries (second to left bar cluster), the ensemble agents outperform the adversaries by a large margin compared to when the roles are reversed (third to left bar cluster).

5.4 Additional Details

Multi-Agent Deep Deterministic Policy Gradient Algorithm

For completeness, we provide the MADDPG algorithm below.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Additional Experimental Details

In all of our experiments, we use the Adam optimizer with a learning rate of 0.01 and $\tau = 0.01$ for updating the target networks. γ is set to be 0.95. The size of the replay buffer is 10^6 and we update the network parameters after every 100 samples added to the replay buffer. We use a batch size of 1024 episodes before making an update, except for TRPO where we found a batch size of 50 lead to better performance (allowing it more updates relative to MADDPG). We train with 10 random seeds for environments with stark success/ fail conditions (cooperative communication, physical deception, and covert communication) and 3 random seeds for the other environments.

The details of the experimental results are shown in the following tables.

Agent $\boldsymbol{\pi}$	**Target reach %**	**Average distance**
MADDPG	**84.0%**	**0.133**
DDPG	32.0%	0.456
DQN	24.8%	0.754
Actor-Critic	17.2%	2.071
TRPO	20.6%	1.573
REINFORCE	13.6%	3.333

Table 5.1: Percentage of episodes where the agent reached the target landmark and average distance from the target in the cooperative communication environment, after 25000 episodes. Note that the percentage of targets reached is different than the policy learning success rate in Figure 5.4b, which indicates the percentage of runs in which the correct policy was learned (consistently reaching the target landmark). Even when the correct behavior is learned, agents occasionally hover slightly outside the target landmark on some episodes, and conversely agents who learn to go to the middle of the landmarks occasionally stumble upon the correct landmark.

	$N = 3$		$N = 6$	
Agent $\boldsymbol{\pi}$	**Average dist.**	**# collisions**	**Average dist.**	**# collisions**
MADDPG	**1.767**	**0.209**	**3.345**	**1.366**
DDPG	1.858	0.375	3.350	1.585

Table 5.2: Average # of collisions per episode and average agent distance from a landmark in the cooperative navigation task, using 2-layer 128 unit MLP policies.

Variance of Policy Gradient Algorithms in a Simple Multi-Agent Setting

To analyze the variance of policy gradient methods in multi-agent settings, we consider a simple cooperative scenario with N agents and binary actions: $P(a_i = 1) = \theta_i$. We define

Agent $\boldsymbol{\pi}$	Adversary $\boldsymbol{\pi}$	**# touches (pp1)**	**# touches (pp2)**
MADDPG	MADDPG	11.0	0.202
MADDPG	DDPG	**16.1**	**0.405**
DDPG	MADDPG	10.3	0.298
DDPG	DDPG	9.4	0.321

Table 5.3: Average number of prey touches by predator per episode on two predator-prey environments with $N = L = 3$, one where the prey (adversaries) are slightly (30%) faster (PP1), and one where they are significantly (100%) faster (PP2). All policies in this experiment are 2-layer 128 unit MLPs.

		$N = 2$			$N = 4$		
Agent $\boldsymbol{\pi}$	Adversary $\boldsymbol{\pi}$	**AG succ %**	**ADV succ %**	**Δ succ %**	**AG succ %**	**ADV succ %**	**Δ succ %**
MADDPG	MADDPG	94.4%	39.2%	55.2%	81.5%	28.3%	**53.2%**
MADDPG	DDPG	92.2%	16.4%	**75.8%**	69.6%	19.8%	49.4%
DDPG	MADDPG	68.9%	59.0%	9.9%	35.7%	32.1%	3.6%
DDPG	DDPG	74.7%	38.6%	36.1%	18.4%	35.8%	-17.4%

Table 5.4: Results on the physical deception task, with $N = 2$ and 4 cooperative agents/landmarks. Success (*succ %*) for agents (AG) and adversaries (ADV) is if they are within a small distance from the target landmark.

Alice, Bob $\boldsymbol{\pi}$	Eve $\boldsymbol{\pi}$	**Bob succ %**	**Eve succ %**	**Δ succ %**
MADDPG	MADDPG	96.5%	52.1%	44.4%
MADDPG	DDPG	96.8%	44.4%	**52.4%**
DDPG	MADDPG	65.3%	64.3%	1.0%
DDPG	DDPG	92.7%	67.6%	25.1%

Table 5.5: Agent (Bob) and adversary (Eve) success rate (*succ %*, i.e. correctly reconstructing the speaker's message) in the covert communication environment. The input message is drawn from a set of two 4-dimensional one-hot vectors.

	S. AG.	*E.* AG.
S. Adv.	7.94	7.74
E. Adv.	8.35	8.11

(a) KA: average frames that the adversary reaches the goal. Adv: the larger the better.

	S. AG.	*E.* AG.
S. Adv.	4.25	4.10
E. Adv.	5.55	4.44

(b) PD: average frames that the adversary stays at the goal. Adv.: the larger the better.

	S. AG.	*E.* AG.
S. Adv.	0.201	0.211
E. Adv.	0.125	0.17

(c) PP: average number of collisions. For Adv., the smaller the better.

Table 5.6: Evaluations of the adversary agent w./w.o. policy ensembles over 1000 trials on different scenarios including (a) keep-away (KA) with $N = M = 1$, (b) physical deception (PD) with $N = 2$ and (c) predator-prey (PP) with $N = 4$ and $L = 1$. *S.* denotes agents with a single policy. *E.* denotes agents with policy ensembles.

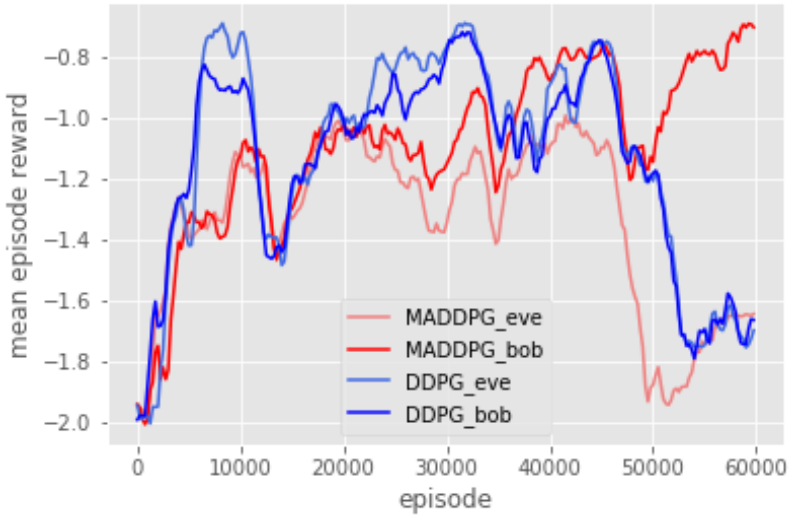


Figure 5.7: In competitive environments such as 'covert communication', the reward can oscillate significantly as agents adapt to each other. DDPG is often unable to overcome this, whereas our MADDPG algorithm has much greater success.

the reward to be 1 if all actions are the same $a_1 = a_2 = \ldots = a_N$, and 0 otherwise. This is a simple scenario with no temporal component: agents must simply learn to either always output 1 or always output 0 at each time step. Despite this, we can show that the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents N.

Proposition 1. *Consider N agents with binary actions: $P(a_i = 1) = \theta_i$, where $R(a_1, \ldots, a_N) = \mathbf{1}_{a_1 = \cdots = a_N}$. We assume an uninformed scenario, in which agents are initialized to $\theta_i = 0.5\ \forall i$. Then, if we are estimating the gradient of the cost J with policy gradient, we have:*

$$P(\langle \hat{\nabla} J, \nabla J \rangle > 0) \propto (0.5)^N,$$

where $\hat{\nabla} J$ is the policy gradient estimator from a single sample, and ∇J is the true gradient.

Proof. We can write $P(a_i) = \theta_i^{a_i}(1 - \theta_i)^{1 - a_i}$, and $\log P(a_i) = a_i \log \theta_i + (1 - a_i) \log(1 - \theta_i)$.

The policy gradient estimator (from a single sample) is:

$$\begin{aligned}\frac{\hat{\partial}}{\partial\theta_i}J &= R(a_1,\ldots,a_N)\frac{\partial}{\partial\theta_i}\log P(a_1,\ldots,a_N)\\ &= R(a_1,\ldots,a_N)\frac{\partial}{\partial\theta_i}\sum_i a_i\log\theta_i + (1-a_i)\log(1-\theta_i)\\ &= R(a_1,\ldots,a_N)\frac{\partial}{\partial\theta_i}(a_i\log\theta_i + (1-a_i)\log(1-\theta_i))\\ &= R(a_1,\ldots,a_N)\left(\frac{a_i}{\theta_i} - \frac{1-a_i}{1-\theta_i}\right)\end{aligned} \tag{5.11}$$

For $\theta_i = 0.5$ we have:

$$\frac{\hat{\partial}}{\partial\theta_i}J = R(a_1,\ldots,a_N)\,(2a_i - 1)$$

And the expected reward can be calculated as:

$$\mathbb{E}(R) = \sum_{a_1,\ldots,a_N} R(a_1,\ldots,a_N)(0.5)^N$$

Consider the case where $R(a_1,\ldots,a_N) = \mathbf{1}_{a_1=\cdots=a_N=1}$. Then

$$\mathbb{E}(R) = (0.5)^N$$

and

$$\mathbb{E}(\frac{\hat{\partial}}{\partial\theta_i}J) = \frac{\partial}{\partial\theta_i}J = (0.5)^N$$

The variance of a single sample of the gradient is then:

$$\mathbb{V}(\frac{\hat{\partial}}{\partial\theta_i}J) = \mathbb{E}(\frac{\hat{\partial}}{\partial\theta_i}J^2) - \mathbb{E}(\frac{\hat{\partial}}{\partial\theta_i}J)^2 = (0.5)^N - (0.5)^{2N}$$

What is the probability of taking a step in the right direction? We can look at $P(\langle\hat{\nabla}J, \nabla J\rangle > 0)$. We have:

$$\langle\hat{\nabla}J, \nabla J\rangle = \sum_i \frac{\hat{\partial}}{\partial\theta_i}J \times (0.5)^N = (0.5)^N\sum_i \frac{\hat{\partial}}{\partial\theta_i}J,$$

so $P(\langle\hat{\nabla}J, \nabla J\rangle > 0) = (0.5)^N$. Thus, as the number of agents increases, the probability of taking a gradient step in the right direction decreases exponentially. □

While this is a somewhat artificial example, it serves to illustrate that there are simple environments that become progressively more difficult (in terms of the probability of taking a gradient step in a direction that increases reward) for policy gradient methods as the number of agents grows. This is particularly true in environments with sparse rewards, such as the

one described above. Note that in this example, the policy gradient variance $\mathbb{V}(\frac{\hat{\partial}}{\partial \theta_i} J)$ actually decreases as N grows. However, the expectation of the policy gradient decreases as well, and the signal to noise ratio $\mathbb{E}(\frac{\hat{\partial}}{\partial \theta_i} J)/(\mathbb{V}(\frac{\hat{\partial}}{\partial \theta_i} J))^{1/2}$ decreases with N, corresponding to the decreasing probability of a correct gradient direction. The intuitive reason a centralized critic helps reduce the variance of the gradients is that we remove a source of uncertainty; conditioned only on the agent's own actions, there is significant variability associated with the actions of other agents, which is largely removed when using these actions as input to the critic.

5.5 Summary

We have proposed a multi-agent policy gradient algorithm where agents learn a centralized critic based on the observations and actions of all agents. Empirically, our method outperforms traditional RL algorithms on a variety of cooperative and competitive multi-agent environments. We can further improve the performance of our method by training agents with an ensemble of policies, an approach we believe to be generally applicable to any multi-agent algorithm.

One downside to our approach is that the input space of Q grows linearly (depending on what information is contained in $\mathbf{x}$) with the number of agents N. This could be remedied in practice by, for example, having a modular Q function that only considers agents in a certain neighborhood of a given agent. We leave this investigation to future work.

Chapter 6

Robust Multi-Agent Reinforcement Learning via Minimax Optimization

Agents trained by DRL tend to be brittle and sensitive to the training environment, especially in the multi-agent scenarios. In the multi-agent setting, a DRL agent's policy can easily get stuck in a poor local optima w.r.t. its training partners – the learned policy may be only locally optimal to other agents' current policies. In this chapter, we focus on the problem of training robust DRL agents with continuous actions in the multi-agent learning setting so that the trained agents can still generalize when its opponents' policies alter. To tackle this problem, we proposed a new algorithm, *MiniMax Multi-agent Deep Deterministic Policy Gradient (M3DDPG)* with the following contributions: (1) we introduce a minimax extension of the popular multi-agent deep deterministic policy gradient algorithm (MADDPG), for robust policy learning; (2) since the continuous action space leads to computational intractability in our minimax learning objective, we propose *Multi-Agent Adversarial Learning (MAAL)* to efficiently solve our proposed formulation. We empirically evaluate our M3DDPG algorithm in four mixed cooperative and competitive multi-agent environments and the agents trained by our method significantly outperforms existing baselines.

6.1 Motivation

Most real-world problems involve interactions between multiple agents and the complexity of problem increases significantly when the agents co-evolve together. Thanks to the recent advances of deep reinforcement learing (DRL) on single agent scenarios, which led to successes in playing Atari game [Mnih et al., 2015], playing go [Silver et al., 2016] and robotics control [Levine et al., 2016], it has been a rising trend to adapt single agent DRL algorithms to multi-agent learning scenarios and many works have shown great successes on a variety of problems, including automatic discovery of communication and language [Sukhbaatar et al., 2016, Mordatch and Abbeel, 2018], multiplayer games [Peng et al., 2017, OpenAI et al., 2019b], traffic control [Wu et al., 2017a] and the analysis of social dilemmas [Leibo et al., 2017].

The critical challenge when adapting classical single agent DRL algorithms to multi-agent setting is the training instability issue: as training progresses, each agent's policy is changing and therefore the environment becomes non-stationary from the perspective of any individual agent (in a way that is not explainable by changes in the agent's own policy). This non-stationary problem can cause significant problems when directly applying the single agent DRL algorithms, for example, the variance of the policy gradient can be exponentially large when the number of agents increases [Lowe et al., 2017]. To handle this instability issue, recent works, such as the counterfactual multi-agent policy gradients [Foerster et al., 2018b] and the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [Lowe et al., 2017], proposed to utilized a *centralized critic* within the actor-critic learning framework to reduce the variance of policy gradient.

Despite the fact that using a centralized critic stabilizes training, the learned policies can still be brittle and sensitive to its training partners and converge to a poor local mode. This is particularly severe for competitive environments: when the opponents alter their policies during testing, the performance of the learned policies can be drastically worse [Lazaridou et al., 2017]. Hence, a *robust* policy becomes desirable in multi-agent setting: a well-trained agent should be able behave well in testing when competing against opponents even with strategies different from its training partners.

In this work, we focus on robust multi-agent reinforcement learning with continuous action spaces and propose a novel algorithm, *MiniMax Multi-agent Deep Deterministic Policy Gradient (M3DDPG)*. M3DDPG is a minimax extension[1] of the classical MADDPG algorithm [Lowe et al., 2017]. Its core idea is that during training, we force each agent to behave well even when its training opponents response in the worst way.

Our major contributions are summarized as follow:

- We introduce the minimax approach to robust multi-agent DRL and propose a novel minimax learning objective based on the MADDPG algorithm;
- In order to efficiently optimize the minimax learning objective, we propose an end-to-end learning approach, *Multi-agent Adversarial Learning (MAAL)*, which is inspired by the *adversarial training* [Goodfellow et al., 2014b] technique[2].
- We empirically evaluate our proposed M3DDPG algorithm on four mixed cooperative and competitive environments and the agents trained by M3DDPG outperform baseline policies on all these environments.

In the rest of the chapter, we will firstly present related works in section 2. Notations and standard algorithms are described in section 3. Our main algorithm, M3DDPG, is introduced in section 4. Experimental results are in section 5.

[1] In fact, we are dealing with gains, i.e., maximizing each agent's accumulative reward, so the "minimax" here is essentially "maximin". We keep the term "minimax" to be consistent with literature.

[2] The connection between MAAL and adversarial training will be discussed in details at the end of section 4.

Related Work

Multi-agent reinforcement learning [Littman, 1994] has been a long-standing field in AI [Hu and Wellman, 1998a, Busoniu et al., 2008]. Recent works in DRL use deep neural networks to approximately represent policy and value functions. Inspired by the success of DRL in single-agent settings, many DRL-based multi-agent learning algorithms have been proposed. Forester et al. [Foerster et al., 2016] and He et al. [He et al., 2016a] extended the deep Q-learning to multi-agent setting; Peng et al. [Peng et al., 2017] proposed a centralized policy learning algorithm based on actor-critic policy gradient; Forester et al. [Foerster et al., 2018b] developed a decentralized multi-agent policy gradient algorithm with centralized baseline; Lowe et al. [Lowe et al., 2017] extended DDPG to multi-agent setting with a centralized Q function; Wei et al. [Wei et al., 2018] and Grau-Moya [Grau-Moya et al., 2018] proposed multi-agent variants of the soft-Q-learning algorithm [Haarnoja et al., 2017]; Yang et al. [Yang et al., 2018] focused on multi-agent reinforcement learning on a very large population of agents. Our M3DDPG algorithm is built on top of MADDPG and inherits the decentralized policy and centralized critic framework.

Minimax is a fundamental concept in game theory and can be applied to general decision-making under uncertainty, prescribing a strategy that minimizes the possible loss for a worst case scenario [Osborne, 2004]. Minimax was firstly introduced to multi-agent reinforcement learning as minimax Q-learning by Littman [Littman, 1994]. More recently, some works combine the minimax framework and the DRL techniques to find Nash equilibrium in two player zero-sum games [Foerster et al., 2018a, Perolat et al., 2017, Grau-Moya et al., 2018]. In our work, we utilize the minimax idea for the purpose of robust policy learning.

Robust reinforcement learning was originally introduced by Morimoto et al. [Morimoto and Doya, 2005] considering the generalization ability of the learned policy in the single-agent setting. This problem is also studied recently with deep neural networks, such as adding random noise to input [Tobin et al., 2017] or dynamics [Peng et al., 2018b] during training. Besides adding random noise, some other works implicitly adopt the minimax idea by utilizing the "worst noise" [Pinto et al., 2017, Mandlekar et al., 2017]. These works force the learned policy to work well even under the worst case perturbations and are typically under the name of "adversarial reinforcement learning", despite the fact that the original adversarial reinforcement learning problem was introduced in the setting of multi-agent learning [Uther and Veloso, 1997]. In our M3DDPG algorithm, we focus on the problem of learning polices that is robust to opponents with different strategies.

Within the minimax framework, finding the worst case scenario is a critical component. Lanctot et al. [Lanctot et al., 2017] proposed an iterative approach that alternatively computes the best response policy while fixes the other. Gao et al. [Gao et al., 2018] replace "mean" in the temporal difference learning rule with "minimum". In our work, we proposed MAAL, which is a general, efficient and fully end-to-end learning approach. MAAL is motivated by adversarial training [Goodfellow et al., 2014b] and suitable for arbitrary number of agents. The core idea of MAAL is approximating the minimization in our min-max objective by a single gradient descent step. The idea of one-step-gradient approximation was also explored

in meta-learning [Finn et al., 2017a].

6.2 Preliminary

In this section, we describe our problem setting and the standard algorithms. Most of the definitions and notations follow the original MADDPG work [Lowe et al., 2017], as described in Chapter 5.

Markov Games

We consider a multi-agent extension of Markov decision processes (MDPs) called partially observable Markov games [Littman, 1994]. A Markov game for N agents is defined by a set of states $\mathcal{S}$ describing the possible configurations of all agents, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agent. To choose actions, each agent i uses a stochastic policy $\boldsymbol{\pi}_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \mapsto [0, 1]$ parameterized by θ_i, which produces the next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$. Each agent i obtains rewards as a function of the state and agent's action $r_i : \mathcal{S} \times \mathcal{A}_i \mapsto \mathbb{R}$, and receives a private observation correlated with the state $\mathbf{o}_i : \mathcal{S} \mapsto \mathcal{O}_i$. The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$. Each agent i aims to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$ where γ is a discount factor and T is the time horizon.

To minimize notation, in the following discussion we will often omit θ from the subscript of $\boldsymbol{\pi}$.

Q-Learning and Deep Q-Networks (DQN)

Q-Learning and DQN [Mnih et al., 2015] are popular methods in reinforcement learning and have been previously applied to multi-agent settings [Foerster et al., 2016, Tesauro, 2004]. Q-Learning makes use of an action-value function for policy $\boldsymbol{\pi}$ as $Q^{\boldsymbol{\pi}}(s, a) = \mathbb{E}[R|s^t = s, a^t = a]$. This Q function can be recursively rewritten as $Q^{\boldsymbol{\pi}}(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \boldsymbol{\pi}}[Q^{\boldsymbol{\pi}}(s', a')]]$. DQN learns the action-value function Q^* corresponding to the optimal policy by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(Q^*(s, a|\theta) - y)^2], \qquad \text{where} \qquad y = r + \gamma \max_{a'} \bar{Q}^*(s', a'). \tag{6.1}$$

$\bar{Q}$ is a target Q function whose parameters are periodically updated with the most recent θ, which helps stabilize learning. Another crucial component of stabilizing DQN is the use of an experience replay buffer $\mathcal{D}$ containing tuples (s, a, r, s'). Q-learning algorithm is most suitable for DRL agents with discrete action spaces.

Policy Gradient (PG) Algorithms

Policy gradient methods is another popular choice for a variety of RL tasks. Let ρ^{π} denote discounted state visitation distribution for a policy π. The main idea of PG is to directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s\sim\rho^{\boldsymbol{\pi}},a\sim\boldsymbol{\pi}_\theta}[R]$ by taking steps in the direction of $\nabla_\theta J(\theta)$. Using the Q function defined previously, the gradient of the policy can be written as [Sutton et al., 2000]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^{\boldsymbol{\pi}},a\sim\boldsymbol{\pi}_\theta}[\nabla_\theta \log \boldsymbol{\pi}_\theta(a|s) Q^{\boldsymbol{\pi}}(s,a)], \tag{6.2}$$

where $p^{\boldsymbol{\pi}}$ is the state distribution. The policy gradient theorem has given rise to several practical algorithms, which often differ in how they estimate $Q^{\boldsymbol{\pi}}$. For example, one can simply use a sample return $R^t = \sum_{i=t}^{T} \gamma^{i-t} r_i$, which leads to the REINFORCE algorithm [Williams, 1992]. Alternatively, one could learn an approximation of the true action-value function $Q^{\boldsymbol{\pi}}(s,a)$ called the *critic* and leads to a variety of *actor-critic* algorithms [Sutton and Barto, 1998].

Deterministic Policy Gradient (DPG) Algorithms

DPG algorithms extends the policy gradient algorithm to deterministic policies $\boldsymbol{\mu}_\theta : \mathcal{S} \mapsto \mathcal{A}$ [Silver et al., 2014]. In particular, under certain conditions we can write the gradient of the objective $J(\theta) = \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}[R(s,a)]$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\nabla_\theta \boldsymbol{\mu}_\theta(s) \nabla_a Q^{\boldsymbol{\mu}}(s,a)|_{a=\boldsymbol{\mu}_\theta(s)}], \tag{6.3}$$

where $\mathcal{D}$ is the replay buffer. Since this theorem relies on $\nabla_a Q^{\boldsymbol{\mu}}(s,a)$, it requires the action space $\mathcal{A}$ (and thus the policy $\boldsymbol{\mu}$) be continuous.

Deep deterministic policy gradient (DDPG) [Lillicrap et al., 2015] is a variant of DPG where the policy $\boldsymbol{\mu}$ and critic $Q^{\boldsymbol{\mu}}$ are approximated with deep neural networks. DDPG is an off-policy algorithm, and samples trajectories from a replay buffer of experiences that are stored throughout training. DDPG also makes use of a target network, as in DQN [Mnih et al., 2015].

Multi-Agent Deep Deterministic Policy Gradient

Directly applying single-agent RL algorithms to the multi-agent setting by treating other agents as part of the environment is problematic as the environment appears non-stationary from the view of any one agent, violating Markov assumptions required for convergence. Particularly, this non-stationary issue is more severe in the case of DRL with neural networks as function approximators. The core idea of the MADDPG algorithm [Lowe et al., 2017] is learning a centralized Q function for each agent which conditions on global information to alleviate the non-stationary problem and stabilize training.

More concretely, consider a game with N agents with policies parameterized by $\boldsymbol{\theta} = \{\theta_1, ..., \theta_N\}$, and let $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_N\}$ be the set of all agents' policies. Then we can write the gradient of the expected return for agent i with policy $\boldsymbol{\mu}_i$, $J(\theta_i) = \mathbb{E}[R_i]$ as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}}[\nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i = \boldsymbol{\mu}_i(o_i)}], \tag{6.4}$$

Here $Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)$ is a *centralized action-value function* that takes as input the actions of all agents, $a_1, \ldots, a_N$, in addition to some state information $\mathbf{x}$ (i.e., $\mathbf{x} = (o_1, ..., o_N)$) , and outputs the Q-value for agent i. Let $\mathbf{x}'$ denote the next state from $\mathbf{x}$ after taking actions $a_1, \ldots, a_N$. The experience replay buffer $\mathcal{D}$ contains the tuples $(\mathbf{x}, \mathbf{x}', a_1, \ldots, a_N, r_1, \ldots, r_N)$, recording experiences of all agents. The centralized action-value function $Q_i^{\boldsymbol{\mu}}$ is updated as:

$$\begin{aligned} \mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \\ y &= r_i + \gamma \, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}', a_1', \ldots, a_N')\big|_{a_j' = \boldsymbol{\mu}_j'(o_j)}, \end{aligned} \tag{6.5}$$

where $\boldsymbol{\mu}' = \{\boldsymbol{\mu}_{\theta_1'}, ..., \boldsymbol{\mu}_{\theta_N'}\}$ is the set of target policies with delayed parameters θ_i'.

Note that the centralized Q function is only used during training. During decentralized execution, each policy $\boldsymbol{\mu}_{\theta_i}$ only takes local information o_i to produce an action.

6.3 Methods

In this section, we introduce our proposed new algorithm, *Minimax Multi-agent Deep Deterministic Policy Gradient (M3DDPG)*, which is built on top of the MADDPG algorithm and particularly designed to improve the *robustness* of learned policies. Our M3DDPG algorithm contains two major novel components:

Minimax Optimization Motivated by the minimax concept in game theory, we introduce minimax optimization into the learning objective;

Multi-Agent Adversarial Learning The continuous action space results in computational intractability issue when optimizing our proposed minimax objective. Hence, we propose *Multi-Agent Adversarial Learning (MAAL)* to solve this optimization problem.

Minimax Optimization

In multi-agent RL, the agents' policies can be very sensitive to their learning partner's policy. Particularly in competitive environments, the learned policies can be brittle when the opponents alter their strategies. For the purpose of learning robust policies, we propose to update policies considering the *worst situation*: during training, we optimize the accumulative reward for each agent i under the assumption that all other agents acts adversarially. This

yields the minimax learning objective $\max_{\theta_i} J_{\mathrm{M}}(\theta_i)$ where

$$\begin{aligned} J_{\mathrm{M}}(\theta_i) &= \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}\left[R_i\right] \\ &= \min_{a^t_{j\neq i}} \mathbb{E}_{s\sim\rho^{\boldsymbol{\mu}}}\left[\sum_{t=0}^{T}\gamma^t\, r_i(s^t,a^t_1,\ldots,a^t_N)\big|_{a^t_i=\boldsymbol{\mu}(o^t_i)}\right] \qquad (6.6)\\ &= \mathbb{E}_{s^0\sim\rho}\left[\min_{a^0_{j\neq i}} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(s^0,a^0_1,\ldots,a^0_N)\big|_{a^0_i=\boldsymbol{\mu}(o^0_i)}\right]. \qquad (6.7)\end{aligned}$$

Critically, in Eq. 6.6, state s^{t+1} at time $t+1$ depends not only on the dynamics $\rho^{\boldsymbol{\mu}}$ and the action $\boldsymbol{\mu}_i(o^t_i)$ but also on all the previous adversarial actions $a^{t'}_{j\neq i}$ with $t'\leq t$. In Eq. 6.7, we derive the modified Q function $Q^{\boldsymbol{\mu}}_{\mathrm{M}}(s,a_1,\ldots,a_N)$, which is naturally centralized and can be rewritten in a recursive form

$$Q^{\boldsymbol{\mu}}_{\mathrm{M},i}\left(s,a_1,\ldots,a_N\right) = r_i(s,a_1,\ldots,a_N) + \gamma\mathbb{E}_{s'}\left[\min_{a'_{j\neq i}} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}\left(s',a'_1,\ldots,a'_N\right)\big|_{a'_i=\boldsymbol{\mu}_i(s')}\right]. \qquad (6.8)$$

Importantly, $Q^{\boldsymbol{\mu}}_{\mathrm{M}}(s,a_1,\ldots,a_N)$ conditions on the current state s as well as the *current actions* $a_1,\ldots,a_N$ and represents the current reward plus the discounted worst case future return starting from the *next state*, s'. This definition brings the benefits that we can naturally apply off-policy temporal difference learning later to derive the update rule for $Q^{\boldsymbol{\mu}}_{\mathrm{M}}$.

Note that for each agent i, none of the adversarial actions depend on its parameter θ_i, so we can directly apply the deterministic policy gradient theorem to compute $\nabla_{\theta_i}J_{\mathrm{M}}(\theta_i)$ and use off-policy temporal difference to update the Q function. Thanks to the *centralized Q function* in MADDPG (Eq. 6.4), which takes in the actions from all the agents, our derivation naturally applies and is perfectly aligned with the MADDPG formulation (Eq. 6.4) by injecting a *minimization* over other agents' actions as follows:

$$\nabla_{\theta_i}J_{\mathrm{M}}(\theta_i) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\begin{array}{l}\nabla_{\theta_i}\boldsymbol{\mu}_i(o_i)\nabla_{a_i}Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x},a^\star_1,\ldots,a_i,\ldots a^\star_N)\Big| \\ \qquad a_i=\boldsymbol{\mu}_i(o_i) \\ \qquad a^\star_{j\neq i}=\arg\min_{a_{j\neq i}} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x},a_1,\ldots,a_N)\end{array}\right], \qquad (6.9)$$

where $\mathcal{D}$ denotes the replay buffer and $\mathbf{x}$ denotes the state information.

Correspondingly, we obtain the new Q function update rule by adding another minimization to Eq. 6.5 when computing the target Q value:

$$\begin{aligned}\mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'\sim\mathcal{D}}[(Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x},a_1,\ldots,a_N)-y)^2], \qquad (6.10)\\ y &= r_i + \gamma\, Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}',{a'^\star}_1,\ldots,a'_i,\ldots,{a'^\star}_N)\\ a'_i &= \boldsymbol{\mu}'_i(o_i),\\ {a'^\star}_{j\neq i} &= \arg\min_{a'_{j\neq i}} Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}',a'_1,\ldots,a'_N),\end{aligned}$$

where $\boldsymbol{\mu}'_i$ denotes the target policy of agent i with delayed parameters θ'_i, and $Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}$ denotes the target Q network for agent i. Combining Eq. 6.9 and Eq. 6.10 yields our proposed minimax learning framework.

Algorithm 2: Minimax Multi-Agent Deep Deterministic Policy Gradient (M3DDPG) for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration, and receive initial state information $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state information $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$, and set $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^k, a^k, r^k, \mathbf{x}'^k)$ from $\mathcal{D}$
      Set $y^k = r_i^k + \gamma\, Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}'^k, a'_1, \ldots, a'_N)|_{a'_i = \boldsymbol{\mu}'_i(o_i^k), a'_{j\neq i} = \boldsymbol{\mu}'_j(o_j^k) + \hat{\epsilon}'_j}$ with $\hat{\epsilon}'_j$ defined in Eq. 6.14
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_k \left(y^k - Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}^k, a_1^k, \ldots, a_N^k)\right)^2$
      Update actor using the sampled policy gradient with $\hat{\epsilon}_j$ defined in Eq. 6.13:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_k \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^k)\nabla_{a_i} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}^k, a_1^\star, \ldots, a_i, \ldots, a_N^\star)\big|_{a_i = \boldsymbol{\mu}_i(o_i^k), a^\star_{j\neq i} = a_j^k + \hat{\epsilon}_j}$$

    end for
    Update target network parameters for each agent i: $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  end for
end for

Multi-Agent Adversarial Learning

The critical challenge in our proposed minimax learning framework is how to handle the embedded minimization in Eq. 6.9 and Eq. 6.10. Due to the continuous action space as well as the non-linearity of Q function, directly optimizing the minimization problem is computationally intractable. A naive approximate solution can be performing an inner-loop gradient descent whenever performing an update step of Eq. 6.9 or Eq. 6.10, but this is too computationally expensive for practical use. Here we introduce an efficient and end-to-end solution, multi-agent adversarial learning (MAAL). The main ideas of MAAL can be summarized in two steps: (1) approximate the non-linear Q function by a locally linear function; (2) replace the inner-loop minimization with a 1-step gradient descent. Note the

core idea of MAAL, locally linearizing the Q function, is adapted from the recent *adversarial training* technique originally developed for supervised learning. We will discuss the connection between adversarial training and MAAL in the end of this section.

For conciseness, we first consider Eq. 6.10 and rewrite it into the following form with auxiliary variables ϵ:

$$\begin{aligned} y &= r_i + \gamma\, Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}', a'^{\star}_1, \ldots, a'_i, \ldots, a'^{\star}_N) \\ a'_k &= \boldsymbol{\mu}'_k(o_k), \;\; \forall 1 \le k \le N \\ a'^{\star}_j &= a'_j + \epsilon_j, \;\; \forall j \neq i \\ \epsilon_{j\neq i} &= \arg\min_{\epsilon_{j\neq i}} Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}', a'_1 + \epsilon_1, \ldots, a'_i, \ldots, a'_N + \epsilon_N). \end{aligned} \tag{6.11}$$

Eq. 6.11 can be interpreted as we are now seeking for a set of *perturbations* ϵ such that the perturbed actions $a'^{\star}$ decrease Q value the most. By linearizing the Q function at $Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a'_1, \ldots, a'_N)$, the desired perturbation ϵ_j can be locally approximated by the gradient direction at $Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a'_1, \ldots, a'_N)$ w.r.t. a'_j. Then we use this local to derive an approximation $\hat{\epsilon}_j$ to the worst case perturbation by taking a small gradient step:

$$\forall j \neq i,\; \hat{\epsilon}_j = -\alpha \nabla_{a_j} Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}', a'_1, \ldots, a_j, \ldots, a'_N), \tag{6.12}$$

where α is a tunable coefficient representing the perturbation rate. It can be also interpreted as the step size of the gradient descent step: when α is too small, the local approximation error will be small but due to the small perturbation, the learned policy can be far from the optimal solution of the minimax objective we proposed; when α is too large, the approximation error may incur too much trouble for the overall learning process and the agents may fail to learn good policies.

We can apply this technique to Eq. 6.9 as well and eventually derive the following formulation:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}} \left[\begin{array}{l} \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i) \nabla_{a_i} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a^{\star}_1, \ldots, a_i, \ldots a^{\star}_N)\Big| \\ \qquad a_i = \boldsymbol{\mu}_i(o_i) \\ \qquad a^{\star}_j = a_j + \hat{\epsilon}_j, \;\; \forall j \neq i \\ \qquad \hat{\epsilon}_j = -\alpha_j \nabla_{a_j} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a_1, \ldots, a_N) \end{array} \right], \tag{6.13}$$

and

$$\begin{aligned} \mathcal{L}(\theta_i) &= \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}[(Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a_1, \ldots, a_N) - y)^2], \\ y &= r_i + \gamma\, Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}', a'^{\star}_1, \ldots, a'_i, \ldots, a'^{\star}_N) \\ a'_k &= \boldsymbol{\mu}'_k(o_k), \;\; \forall 1 \le k \le N \\ a'^{\star}_j &= a'_j + \hat{\epsilon}'_j, \;\; \forall j \neq i \\ \hat{\epsilon}'_j &= -\alpha_j \nabla_{a'_j} Q^{\boldsymbol{\mu}'}_{\mathrm{M},i}(\mathbf{x}, a'_1, \ldots, a'_N), \end{aligned} \tag{6.14}$$

where $\alpha_1, \ldots, \alpha_N$ are additional parameters. MAAL only requires one additional gradient computation, and can be executed in a fully end-to-end fashion. Finally, combining Eq. 6.13 and Eq. 6.14 completes MAAL. The overall algorithm, M3DDPG, is summarized as Algo. 2.

Discussion

Connection to Adversarial Training

Adversarial training is a robust training approach for deep neural networks on supervised learning [Goodfellow et al., 2014a]. The core idea is to force the classifier to predict correctly even when given *adversarial examples*, which are obtained by adding a small adversarial perturbation to the original input data such that the classification loss can be decreased the most. Formally, suppose the classification loss function is $\mathcal{L}(\theta) = \mathbb{E}_{x,y}\left[f_\theta(x; y)\right]$ with input data x and label y. Adversarial training aims to optimize the following adversarial loss instead

$$\begin{aligned} \mathcal{L}_{\text{adv}}(\theta) &= \mathbb{E}_{x,y}\left[f_\theta(x + \epsilon^\star; y)\right] \\ \epsilon^\star &= \arg\max_{\|\epsilon\| \le \alpha} f_\theta(x + \epsilon; y). \end{aligned} \tag{6.15}$$

The core technique to efficiently optimize $\mathcal{L}_{\text{adv}}(\theta)$ is to locally linearize the loss function at $f_\theta(x; y)$ and approximate $\epsilon^\star$ by the scaled gradient.

Thanks to the centralized Q function, which takes the actions from all the agents as part of the input, we are able to easily inject the minimax optimization (Eq. 6.11) and represent it in a similar way to adversarial training (Eq. 6.15) so that we can adopt the similar technique to effectively solve our minimax optimization in a fully end-to-end fashion.

Connection to Single Agent Robust RL

M3DDPG with MAAL can be also viewed as the special case of robust reinforcement learning (RRL) [Morimoto and Doya, 2005] in the single agent setting, which aims to bridge the gap between training in simulation and testing in the real world by adding adversarial perturbations to the transition dynamics during training. Here, we consider the multi-agent setting and add worst case perturbations to *actions* of opponent agents during training. Note that in the perspective of a single agent, perturbations on opponents' actions can be also considered as a special adversarial noise on the dynamics.

Choice of α

In the extreme case of $\alpha = 0$, M3DDPG degenerates to the original MADDPG algorithm while as α increases, the policy learning tends to be more robust but the optimization becomes harder. In practice, using a fixed α throughout training can lead to very unstable learning behavior due to the changing scale of the gradients. The original adversarial training paper [Goodfellow et al., 2014b] suggests to compute ϵ with a fixed norm, namely $g = \nabla_x f_\theta(x; y)$, $\hat{\epsilon} = \alpha \frac{g}{\|g\|}$, where x denotes the input data to the classifier and y denotes the label. Accordingly, in our M3DDPG algorithm, we can adaptively compute the perturbation $\hat{\epsilon}_j$ by

$$g = \nabla_{a_j} Q^{\boldsymbol{\mu}}_{\text{M},i}(\mathbf{x}, a_1, \ldots, a_N), \quad \hat{\epsilon}_j = -\alpha_j \frac{g}{\|g\|}. \tag{6.16}$$

Eq. 6.16 generally works fine in practice but in some hard multi-agent learning environments, unstable training behavior can be still observed. We suspect that it is because the changing norm of actions in these situations. Different from the supervised learning setting where the norm of the input data x is typically stable, in reinforcement learning the norm of actions can drastically change even in a single episode. Therefore, it is possible to see cases that even a perturbation with a small fixed norm overwhelms the action a_j, which may potentially lead to computational stability issue. Therefore, we also introduce the following alternative for adaptive perturbation computation:

$$g = \nabla_{a_j} Q^{\boldsymbol{\mu}}_{\mathrm{M},i}(\mathbf{x}, a_1, \ldots, a_N), \quad \hat{\epsilon}_j = -\alpha_j \|a_j\| \frac{g}{\|g\|}. \tag{6.17}$$

Lastly, note that in a mixed cooperative and competitive environment, ideally we only need to add adversarial perturbations to competitors. But empirically we observe that also adding (smaller) perturbations to collaborators can further improve the quality of learned policies.

6.4 Experiments

We adopt the same *particle-world* environments as the MADDPG work [Lowe et al., 2017] (Chapter 5) as well as the training configurations. α is selected from a grid search over 0.1, 0.01 and 0.001. For testing, we generate a fixed set of 2500 environment configurations (i.e., landmarks and birthplaces) and evaluate on this fixed set for a fair comparison.

The source code is publicly available at: `https://github.com/dadadidodi/m3ddpg`

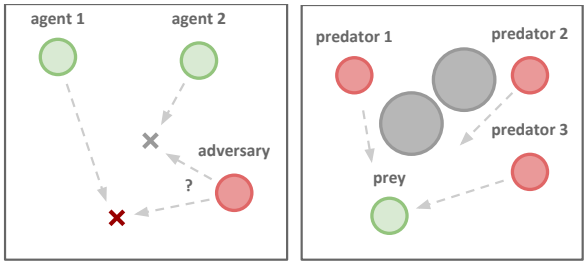


Figure 6.1: Illustrations of some environments we consider, including *Physical Deception* (left) and *Predator-Prey* (right).

Environments

The particle world environment consists of N cooperative agents, M adversarial agents and L landmarks in a two-dimensional world with continuous space. We focus on the four mixed cooperative and competitive scenarios to best examine the effectiveness of our minimax formulation.

Covert communication

This is an adversarial communication environment, where a speaker agent ('Alice') must communicate a message to a listener agent ('Bob') ($N = 2$), who must reconstruct the message at the other end. However, an adversarial agent ('Eve') ($M = 1$) is also observing the channel, and wants to reconstruct the message — Alice and Bob are penalized based on Eve's reconstruction, and thus Alice must encode her message using a randomly generated *key*, known only to Alice and Bob.

Keep-away

This scenario consists of $L = 1$ target landmark, $N = 2$ cooperative agents and $M = 1$ adversarial agent. Cooperating agents need to reach the landmark and keep the adversarial agent away from the landmark by pushing it while the adversarial agent must stay at the landmark to occupy it.

Physical deception

Here, $N = 2$ agents cooperate to reach a single target landmark from a total of $L = 2$ landmarks. They are rewarded based on the minimum distance of any agent to the target (so only one agent needs to reach the target landmark). However, a lone adversary ($M = 1$) also desires to reach the target landmark; the catch is that the adversary does not know which of the landmarks is the correct one. Thus the cooperating agents, who are penalized based on the adversary distance to the target, learn to spread out and cover all landmarks so as to deceive the adversary.

Predator-prey

In this variant of the classic predator-prey game, $N = 3$ slower cooperating agents must chase the faster adversary ($M = 1$) around a randomly generated environment with $L = 2$ large landmarks impeding the way. Each time the cooperative agents collide with an adversary, the agents are rewarded while the adversary is penalized.

Comparison to MADDPG

To evaluate the quality of learned policies trained by different algorithms in competitive scenarios, we measure the performance of agents trained by our M3DDPG and agents by classical MADDPG in the roles of both normal agent and adversary in each environment.

The results are demonstrated in Figure 6.2, where we measure the rewards of the normal agents in different scenarios and normalize them to 0-1. We notice that in all the environments, the highest score is achieved when the M3DDPG agents play as the normal agents against the MADDPG adversary (Minimax vs MA); while the lowest score is when the MADDPG agents

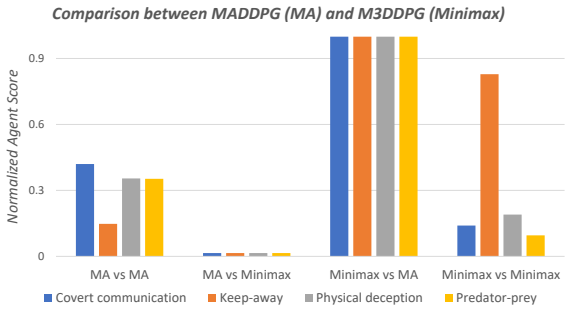


Figure 6.2: Comparison between M3DDPG (Minimax) and classical MADDPG (MA) on the four mixed competitive environments. Each bar cluster shows the 0-1 normalized score for a set of competing policies in different roles (agent vs adversary), where a higher score is better for the agent. In all cases, M3DDPG outperforms MADDPG when directly pitted against it.

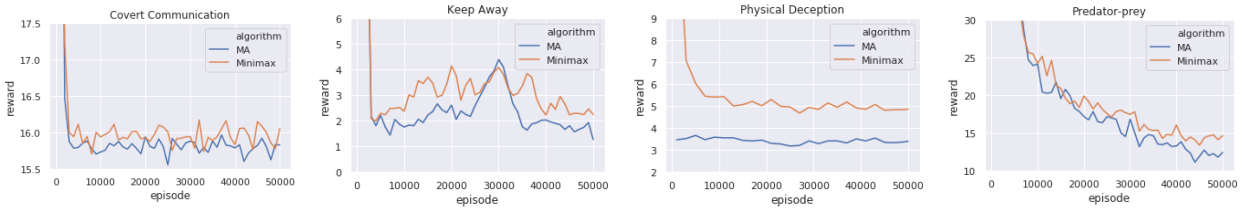


Figure 6.3: Performances of M3DDPG (Minimax, red) and MADDPG (MA, blue) under the *worst situation*, i.e., against the *disruptive adversaries*, on convert communication, keep-away, physical deception and predator-pray from left to right. The y-axis denotes the reward of normal agents (fixed) and x-axis denotes the training episodes performed of the disruptive adversaries. Higher reward implies a more robust policy. Agents trained by M3DDPG (Minimax) perform better on all the scenarios.

act as normal agents against the M3DDPG adversary (MA vs Minimax). This indicates that policies trained by M3DDPG have much higher quality than original MADDPG.

Evaluation with Disruptive Adversaries

Instead of M3DDPG and MADDPG directly competing against each other, now we consider their performances in the *worst* possible situations through their individual competitions against *disruptive adversaries*. We construct disruptive adversaries by (1) fixing the trained normal agents (M3DDPG or MADDPG); (2) setting the reward of the disruptive adversary as the negative value of normal agents' reward (so that the problem becomes zero-sum); and then (3) using DDPG to train the disruptive adversary from scratch under the zero-sum reward setting.

In the particle world environment, the competitive scenarios are generally not zero-sum, which implies that according to the default reward function, the adversaries may have different purposes rather than directly disrupting the normal agents. So, in order to evaluate the

effectiveness our minimax optimization in the worst situation, we convert every problems into a zero-sum form and compare the performances of our M3DDPG agents as well as the MADDPG agents against this artificially constructed adversaries. Moreover, since each of our four environments has only 1 adversary, after fixing the normal agents, the learning problem degenerates to the single agent setting and classical DDPG is sufficient to stably train a disruptive adversary.

The results are shown in Figure 6.3, where we plot the reward of the fixed normal agents of different algorithms as the training of the disruptive adversaries progresses until convergence. Note that due to the different environment designs, the difficulty for the disruptive agents to break the strategy of normal agents varies: for example, in *convert communication*, since the private key is not accessible to the adversary agent, breaking the encrypted communication will be very hard; while in *physical deception*, since we do not allow communication and fix the normal agents, a smart enough adversary may easily infer the target landmark by observing the initial behavior of the two cooperative agents. Nevertheless, despite these intrinsic properties, the M3DDPG agents (Minimax) achieves higher reward in all the scenarios, which implies better robustness even in the worst situation.

6.5 Summary

In this chapter, we propose a novel algorithm, minimax mulit-agent deep deterministic policy gradient (M3DDPG), for robust multi-agent reinforcement learning, which leverages the minimax concept and introduces a minimax learning objective. To efficiently optimize the minimax objective, we propose MAAL, which approximates the inner-loop minimization by a single gradient descent step. Empirically, M3DDPG outperforms the benchmark methods on four mixed cooperative and competitive scenarios.

Nevertheless, due to the single step gradient approximation in MAAL, which is efficient in computation, an M3DDPG agent can only explore locally worst situation during the evolving process at training, which can still lead to unsatisfying behavior when testing opponents have drastically different strategies. It will be an interesting direction to re-examine the robustness-efficiency trade-off in MAAL and further improve policy learning by placing more computations on the minimax optimization. We leave this as our future work.

Part IV

Applications in Natural Language Processing

Chapter 7

Robust Neural Relation Extraction

Relation extraction is one of the core tasks in natural language processing, which aims to extract '*(entity, entity, relation)*' triplets from texts[1]. In this chapter, we introduce the adversarial optimization technique from Chapter 6 to the relation extraction problem, which we previously applied in multi-agent to improve the policy generalization capability. We empirically show that by applying adversarial training to a neural relation extractor in the supervised learning setting, the generalization capability, i.e., test performance, can be significantly boosted.

7.1 Motivation

Despite the recent successes of deep neural networks on various applications, neural network models tend to be overconfident about the noise in input signals. *Adversarial examples* [Szegedy et al., 2013] are examples generated by adding noise in the form of small perturbations to the original data, which are often indistinguishable for humans but drastically increase the loss incurred in a deep model. *Adversarial training* [Goodfellow et al., 2014b] is a technique for regularizing deep models by encouraging the neural network to correctly classify both unmodified examples and perturbed ones, which in practice not only enhances the robustness of the neural network but also improves its generalizability. Previous work has largely applied adversarial training on straightforward classification tasks, including image classification [Goodfellow et al., 2014b] and text classification [Miyato et al., 2017], where the goal is simply predicting a single label for every example and the training examples are able to provide strong supervision. It remains unclear whether adversarial training could be still effective for tasks with much weaker supervision, e.g., distant supervision [Mintz et al., 2009], or a different evaluation metric other than prediction accuracy (e.g., F1 score).

This chapter focuses on the task of *relation extraction*, where the goal is to predict the relation that exists between a particular entity pair given several text mentions. One popular way to handle this problem is the multi-instance multi-label learning framework

[1]For further explanations, readers can refer to [Russell and Norvig, 2016].

(MIML) [Hoffmann et al., 2011, Surdeanu et al., 2012] with distant supervision [Mintz et al., 2009], where the mentions for an entity pair are aligned with the relations in Freebase [Bollacker et al., 2008]. In this setting, relation extraction is much harder than the canonical classification problem in two respects: (1) although distant supervision can provide a large amount of data, the training labels are very noisy, and due to the multi-instance framework, the supervision is much weaker; (2) the evaluation metric of relation extraction is often the precision-recall curve or F1 score, which cannot be represented (and thereby optimized) directly in the loss function.

In order to evaluate the effectiveness of adversarial training for relation extraction, we apply it to two different architectures (a convoluational neural network and a recurrent neural network) on two different datasets. Experimental results show that even on this harder task with much weaker supervision, adversarial training can still improve the performance on all of the cases we studied.

Related Work

Neural Relation Extraction: In recent years, neural network models have shown superior performance over approaches using hand-crafted features in various tasks. Convolutional neural networks (CNN) are among the first deep models that have been applied to relation extraction [dos Santos et al., 2015, Nguyen and Grishman, 2015]. Variants of convolutional networks include piecewise-CNN (PCNN) [Zeng et al., 2014], split CNN [Adel et al., 2016], CNN with sentence-wise pooling [Jiang et al., 2016] and attention CNN [Wang et al., 2016]. Recurrent neural networks (RNN) are another popular choice, and have been used in recent work in the form of recurrent CNNs [Cai et al., 2016] and attention RNNs [Zhou et al., 2016]. An instance-level selective attention mechanism was introduced for MIML by [Lin et al., 2016], and has significantly improved the prediction accuracy for several of these base deep models.

Adversarial Training: Adversarial training (AT) [Goodfellow et al., 2014b] was originally introduced in the context of image classification tasks where the input data is continuous. [Miyato et al., 2015, Miyato et al., 2017] adapts AT to text classification by adding perturbations on word embeddings and also extends AT to a semi-supervised setting by minimizing the entropy of the predicted label distributions on unlabeled data.

AT introduces an end-to-end and deterministic way of data perturbation by utilizing the gradient information. There are also other works for regularizing classifiers by adding random noise to the data, such as dropout [Srivastava et al., 2014] and its variant for NLP tasks, word dropout [Iyyer et al., 2015]. [Xie et al., 2017] discusses various data noising techniques for language models. [Søgaard, 2013] and [Li et al., 2017b] focus on linguistic adversaries.

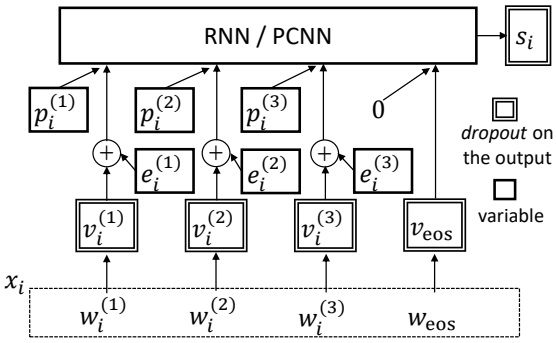


Figure 7.1: The computation graph of encoding a sentence x_i with adversarial training. e_i denotes the adversarial perturbation w.r.t. x_i. Dropout is placed on the output of the variables in the double-lined rectangles.

7.2 Methodology

We first introduce MIML and then describe the base neural network models we consider:[2] piecewise CNN [Zeng et al., 2015] (PCNN) and bidirectional GRU [Cho et al., 2014] (RNN). We also utilize the *selective attention* mechanism in [Lin et al., 2016] for both PCNN and RNN models. Adversarial training is presented at the end of this section.

Preliminaries

In MIML, we consider the set of text sentences $X = \{x_1, x_2, \ldots, x_n\}$ for each entity pair. Supposing we have R predefined relations (including NA) to extract, we want to predict the probability of each of the R relations given the mentions. Formally, for each relation r, we want to predict $P(r \mid x_1, \ldots, x_n)$.

Note that since an entity pair may have no relations, we introduce a special relation NA to the label set. Hence, we simply assume there will be *at least* one relation existing for every entity pair. During evaluation, we ignore the probability predicted for the NA relation.

Neural Architectures

Input Representation: For each sentence x_i, we use pretrained word embeddings to project each word token into d_w-dimensional space. Note that we also need to include the entity position information in x_i. Here we introduce an extra feature vector $p_i^{(w)}$ for each word w to encode the entities' positions. One choice is the *position embedding* [Zeng et al., 2014]: for each word w, we compute the relative distances to the two entities and embed the distances in two d_p-dimensional vectors, which are then concatenated as $p_i^{(w)}$. Position embedding introduces extra variables in the model and slows down the training time. We also investigate a simpler choice, *indicator encoding*: when a word w is exactly an entity, we generate a

[2]We primarily focus on effectiveness of AT. Other techniques in Sec. 7.1 are complementary to our focus.

d_p-dimensional $\vec{\mathbf{1}}$ vector and a $\vec{\mathbf{0}}$ vector otherwise. In our experiments, position embedding is crucial for PCNN due to the spatial invariance of CNN. For RNN, position embedding helps little (likely because an RNN has the capacity of exploiting temporal dependencies) so we adopt indicator encoding instead.

Sentence Encoder: For a sentence x_i, we want to apply a non-linear transformation to the vector representation of x_i to derive a feature vector $s_i = f(x_i; \theta)$ given a set of parameters θ. We consider both PCNN and RNN as $f(x_i; \theta)$.

For PCNN, inheriting the settings from [Zeng et al., 2014], we adopt a convolution kernel with window size 3 and d_s output channels and then apply piecewise pooling and ReLU [Nair and Hinton, 2010] as an activation function to eventually obtain a $3 \cdot d_s$-dimensional feature vector s_i.

For RNN, we adopt bidirectional GRU with d_s hidden units and concatenate the hidden states of the last timesteps from both the forward and the backward RNN as a $2 \cdot d_s$-dimensional feature vector s_i.

Selective Attention: Following [Lin et al., 2016], for each relation r, we aim to softly select an attended sentence s_r by taking a weighted average of $s_1, s_2, \ldots, s_n$, namely $s_r = \sum_i \alpha_i^r s_i$. Here α^r denotes the attention weights w.r.t. relation r. For computing the weights, we define a *query vector* q_r for each relation r and compute $\alpha^r = \text{softmax}(u^r)$ where $u_i^r = \tanh(s_i)^\top q_r$. The query vector q_r can be considered as the embedding vector for the relation r, which is jointly learned with other model parameters.

Loss Function: For an entity pair, we compute the probability of relation r by $P(r \mid X; \theta) = \text{softmax}(As_r + b)$, where A is the projection matrix and b is the bias. For the multi-label setting, suppose K relations $r_1, \ldots, r_K$ exist for X. Simply taking the summation over the log probabilities of all those labels yields the final loss function

$$L(X; \theta) = -\sum_{i=1}^{K} \log P(r_i \mid X; \theta). \tag{7.1}$$

Dropout: For regularizing the parameters, we apply dropout [Srivastava et al., 2014] to both the word embedding and the sentence feature vector s_i. Note that we do not perform dropout on the position embedding p_i.

Adversarial Training

Adversarial training (AT) is a way of regularizing the classifier to improve robustness to small *worst-case* perturbations by computing the gradient direction of a loss function w.r.t. the data. AT generates continuous perturbations, so we add the adversarial noise at the level of the word embeddings, similar to [Miyato et al., 2017]. Formally, consider the input data X and suppose the word embedding of all the words in X is V. AT adds a small adversarial perturbation e_{adv} to V and optimizes the following objective instead of Eq.(7.1).

$$L_{\text{adv}}(X; \theta) = L(X + e_{\text{adv}}; \theta), \text{ where} \tag{7.2}$$

$$e_{\text{adv}} = \arg\max_{\|e\| \le \epsilon} L(X + e; \hat{\theta}) \tag{7.3}$$

Dataset	#Rel	#Ent-Pair	#Mention	Sent-Len
NYT-Train	58	290429	577434	145
UW-Train	5	132419	546731	120

Table 7.1: Dataset statistics (#Rel includes *NA*).

Here $\hat{\theta}$ denotes a fixed copy of the current value of θ. Since Eq.(7.3) is computationally intractable for neural nets, [Goodfellow et al., 2014b] proposes to approximate Eq.(7.3) by linearizing $L(X;\hat{\theta})$ near X:

$$e_{\text{adv}} = \epsilon g/\|g\|, \text{ where } g = \nabla_V L(X;\hat{\theta}). \tag{7.4}$$

Here V denotes the word embedding of *all* the words in X. Accordingly, in Eq. 7.4, $\|g\|$ denotes the norm of gradients over *all* the words from *all* the sentences in X. In addition, we do not perturb the feature vector p for entity positions. A visualization of the process is demonstrated in Fig. 7.1.

7.3 Experiments

To measure the effectiveness of adversarial training on relation extraction, we evaluate both the CNN (PCNN) and RNN (bi-GRU) models on two different datasets, the NYT dataset (NYT) developed by [Riedel et al., 2010] and the UW dataset (UW) by [Liu et al., 2016]. All code is implemented in Tensorflow [Abadi et al., 2016] and available at `https://github.com/jxwuyi/AtNRE`. We adopt Adam optimizer [Kingma and Ba, 2014] with learning rate 0.001, batch size 50 and dropout rate 0.5. For adversarial training, the only parameter is ϵ. In each of the following experiments, we fixed all the hyper-parameters of the base model, performed a binary search solely on ϵ and showed the most effective value of ϵ.

Datasets

The statistics of the two datasets are summarized in Table 7.1. We exclude sentences longer than *Sent-Len* during training and randomly split data for entity pairs with more than 500 mentions. Note that the number of target relations in these two datasets are significantly different, which helps demonstrate the applicability of adversarial training on various evaluation settings.

Since the test set of the UW dataset only contains 200 sentences, we adopt a subset of the test set from the NYT dataset: all the entity pairs with the corresponding 4 relations in UW and another 1500 randomly selected *NA* pairs.

Recall	0.1	0.2	0.3	0.4	AUC
PCNN	0.667	0.572	0.476	0.392	0.329
PCNN-Adv	0.717	0.589	0.511	0.407	0.356
RNN	0.668	0.586	0.524	0.442	0.351
RNN-Adv	**0.728**	**0.646**	**0.553**	**0.481**	**0.382**

Table 7.2: Precisions of various models for different recalls on the NYT dataset, with best values in bold.

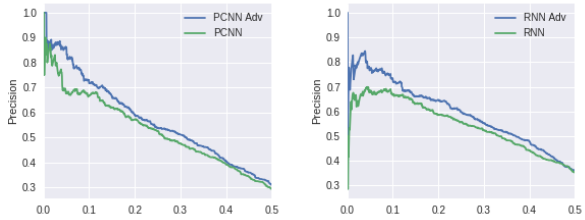


Figure 7.2: PR curves for PCNN (left) and RNN (right) on the NYT dataset with (blue) and without (green) adversarial training.

Practical Performances

The NYT dataset:

We utilize the word embeddings released by [Lin et al., 2016], which has $d_w = 50$ dimensions. For model parameters, we set $d_e = 5$ (dimension of the entity position feature vector) and $d_s = 230$ (dimension of sentence feature vector) for PCNN and $d_e = 3$ and $d_s = 150$ for RNN. For adversarial training, we choose $\epsilon = 0.01$ for PCNN and $\epsilon = 0.02$ for RNN. We empirically observed that when adding dropout to the word embeddings, PCNN performs significantly worse. Hence we only apply dropout to s_i for PCNN. However, even with a dropout rate of 0.5, RNN still performs well. We conjecture that it is due to PCNN being more sensitive to input signals and the dimensionality of the word embedding ($d_w = 50$) being very small.

The precision-recall curves for different models on the test set are shown in Fig. 7.2. Since the precision drops significantly with large recalls on the NYT dataset, we emphasize a part of the curve with recall number smaller than 0.5 in the figure. Adversarial training significantly improves the precision for both PCNN and RNN models. We also show the precision numbers for some particular recalls as well as the AUC (for the whole PR curve) in Table 7.2, where RNN generally leads to better precision.

The UW dataset:

We train a word embedding of $d_w = 200$ dimensions using Glove [Pennington et al., 2014] on the New York Times Corpus in this experiment. For model parameters, we set the entity feature dimension $d_e = 5$ and sentence feature dimension $d_s = 250$ for PCNN and $d_e = 3$ and

Recall	0.1	0.2	0.3	0.4	AUC
PCNN	0.765	0.717	0.713	0.677	0.576
PCNN-Adv	0.844	0.750	0.738	0.707	0.619
RNN	0.823	0.822	0.791	0.752	0.631
RNN-Adv	**0.929**	**0.878**	**0.850**	**0.779**	**0.671**

Table 7.3: Precisions of various models for different recalls on the UW dataset, with best values in bold.

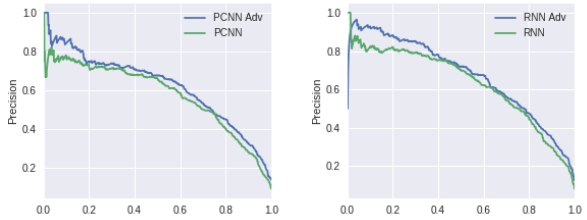


Figure 7.3: PR curves for PCNN (left) and RNN (right) on the UW dataset with (blue) and without (green) adversarial training.

$d_s = 200$ for RNN. For adversarial training, we choose $\epsilon = 0.05$ for PCNN and $\epsilon = 0.5$ for RNN. Since here word embedding dimension d_w is larger than that used for the NYT dataset, which implies that we now have word embeddings with larger norms, accordingly the optimal value of ϵ increases. The precision-recall curves on the test data are shown in Fig. 7.3, where adversarial training again significantly improves the precision for both models. The precision numbers for some particular recall values as well as the AUC numbers are demonstrated in Table 7.3. Similarly RNN yields superior performances on the UW dataset.

Discussion

CNN vs RNN: In the experiments, RNN generally produces more precise predictions than CNN due to its rich model capacity and also has high robustness to input embeddings. The CNN, in contrast, has far fewer parameters which leads to much faster training and testing, which suggests a practical trade-off.

Notably, although the improvement under AUC by adversarial training are roughly the same for both RNN and CNN, the optimal ϵ value for RNN is always much larger than CNN. This implies that empirically RNN is more robust under adversarial attacks than CNN, which also helps RNN maintain higher precision as recall increases.

Choice of ϵ: When $\epsilon = 0$, the AT loss (Eq.(7.2)) degenerates to the original loss (Eq.(7.1)); when ϵ becomes too large, the noise can change the semantics of a sentence[3] and make the

[3]When ϵ is large enough, e.g., comparable to the norm of input embeddings, and if we add the perturbation to word w and consider its nearest unperturbed word embedding in the embedding space, the nearest word will be different from the original word w. This implicitly changes the content of a sentence.

model extremely hard to correctly classify the adversarial examples.

Notably, the optimal value of ϵ is much smaller than the norm of the word embedding, which implies adversarial training works most effectively when only producing tiny perturbations on word features while keeping the semantics of sentences unchanged[4].

Connection to other approaches: [Li et al., 2017b, Xie et al., 2017] proposes linguistic adversaries techniques to enhance the robustness of the model by randomly changing the word tokens in a sentence. This explicitly modifies the semantics of a sentence. By contrast, adversarial training focuses on smaller and continuous perturbations in the embedding space while preserving the semantics of sentences. Hence, adversarial training is complementary to linguistic adversaries.

7.4 Summary

In this chapter, we apply the adversarial training technique, which is a regularization technique for training robust agents, to neural relation extraction. We experiment on a variety of architecture and datasets and demonstrate that our proposed framework generally improves the test accuracy of the trained neural extractor. This indicates a convincing direction to potentially improve test performances on an even wider range of NLP tasks almost for free by simply leveraging the techniques for training robust agents.

[4]The nearest neighbor of word w remains unchanged after w being perturbed with the optimal ϵ.

Chapter 8

Meta-Learning MCMC Proposals for Named Entity Recognition

In natural language processing (NLP), there is an important family of methods which first learns a *probabilistic model* for test sequences from training data and then performs *sampling-based probabilistic inference* on test texts to accomplish a desired NLP task. For high quality probabilistic inference, manually constructed, model-specific proposals are often required. Inspired by recent progresses in meta-learning for training learning agents that can generalize to unseen environments, we propose a meta-learning approach to building effective and generalizable MCMC proposals. We parametrize the proposal as a neural network to provide fast approximations to block Gibbs conditionals. The learned neural proposals generalize to occurrences of common structural motifs across different models, allowing for the construction of a library of learned inference primitives that can accelerate inference on unseen models with no model-specific training required. We first evaluate our approach on several small scale models, in which our learned proposals outperform a hand-tuned sampler. Finally, we apply our method on a real-world NLP task, named entity recognition, in which our sampler yields higher final F1 scores than classical single-site Gibbs sampling.

8.1 Motivation

Model-based probabilistic inference is a highly successful paradigm for machine learning, with applications to tasks as diverse as movie recommendation [Stern et al., 2009], visual scene perception [Kulkarni et al., 2015], music transcription [Berg-Kirkpatrick et al., 2014], *etc.* . People learn and plan using mental models, and indeed the entire enterprise of modern science can be viewed as constructing a sophisticated hierarchy of models of physical, mental, and social phenomena. *Probabilistic programming* provides a formal representation of models as sample-generating programs, promising the ability to explore a even richer range of models. Probabilistic programming language based approaches have been successfully applied to complex real-world tasks such as seismic monitoring [Moore and Russell, 2017], concept

learning [Lake et al., 2015] and design generation [Ritchie et al., 2015].

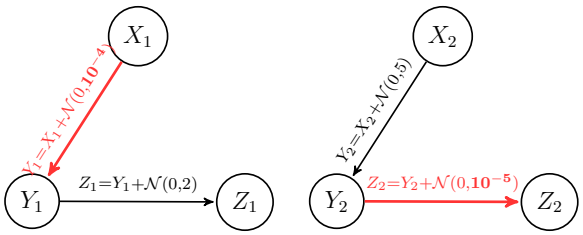
However, most of these applications require *manually* designed proposal distributions for efficient MCMC inference. Commonly used "black-box" MCMC algorithms are often far from satisfactory when handling complex models. Hamiltonian Monte Carlo [Neal, 2011] takes global steps but is only applicable to continuous latent variables with differentiable likelihoods. Single-site Gibbs sampling [Spiegelhalter et al., 1996, Andrieu et al., 2003] can be applied to many model but suffers from slow mixing when variables are coupled in the posterior. Effective real-world inference often requires *block proposals* that update multiple variables together to overcome near-deterministic and long-range dependence structures. However, computing exact Gibbs proposals for large blocks quickly becomes intractable (approaching the difficulty of posterior inference), and in practice it is common to invest significant effort in hand-engineering computational tricks for a particular model.

Can we build tractable MCMC proposals that are (1) effective for fast mixing and (2) ready to be reused across different models?

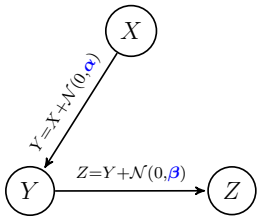
Recent advances in *meta-learning* demonstrate promising results in learning to build reinforcement learning agents that can generalize to unseen environments [Duan et al., 2016b, Tobin et al., 2017, Finn et al., 2017a, Wu et al., 2018]. The core idea of meta-learning is to generate a large number of related training environments under the same objective and then train a learning agent to succeed in all of them. Inspired by those meta-learning works, we can adopt a similar approach to build *generalizable* MCMC proposals.

We propose to learn approximate block-Gibbs proposals that can be reused within a given model, and even across models containing similar *structural motifs* (i.e. , common structural patterns). Recent work recognized that a wide range of models can be represented as compositions of simple components [Grosse et al., 2012], and that domain-specific models may still reuse general structural motifs such as chains, grids, rings, or trees [Kemp and Tenenbaum, 2008]. We exploit this by training a meta-proposal to approximate block-Gibbs conditionals for models containing a given motif, with the model parameters provided as an additional input. At a high level, approach first (1) generates different instantiations of a particular motif by randomizing its model parameters, and then (2) meta-train a neural proposal "close to" the true Gibbs conditionals for all the instantiations (see Fig. 8.1). By learning such flexible samplers, we can improve inference not only within a specific model but even on unseen models containing similar structures, with no additional training required. In contrast to techniques that compile inference procedures specific to a given model [Stuhlmüller et al., 2013, Le et al., 2017, Song et al., 2017a], learning inference artifacts that generalize to novel models is valuable in allowing model builders to quickly explore a wide range of possible models.

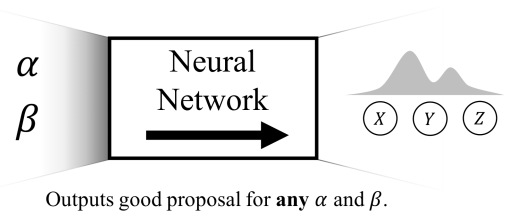
We explore the application of our approach to a wide range of models. On grid-structured models from a UAI inference competition, our learned proposal significantly outperforms Gibbs sampling. For open-universe Gaussian mixture models, we show that a simple learned block proposal yields performance comparable to a model-specific hand-tuned sampler, and generalizes to models more than those it was trained on. We additionally apply our method to a named entity recognition (NER) task, showing that not only do our learned block proposals



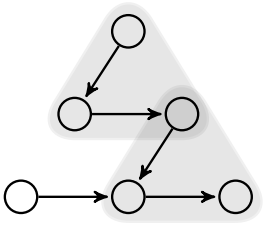
(a) Two models of same structure, but with different *parameters* and thus different near-deterministic relations (shown in red). Naive MCMC algorithms like single-site Gibbs fail on both models due to these dependencies.



(b) To design a single proposal that works on both models in Fig. 8.1a, we consider this *general* model with variable parameters α and β (shown in blue).



(c) Our neural proposal takes model parameters α and β as input, and is trained to output good proposal distributions on randomly generated parameters. Therefore, it performs well for any given α and β. (For simplicity, inputs in diagram omit possible other nodes that the proposed nodes may depend on.)



(d) The neural proposal can be applied anywhere this structural pattern is present (or *instantiated*). The grey regions show example instantiations in this large model. (There are more.)

Figure 8.1: **Toy example:** Naive MCMC algorithms (e.g. , single-site Gibbs) fail when variables are tightly coupled, requiring custom proposals even for models with similar structure but different dependency relations (Fig. 8.1a). Our goal is to design a single proposal that works on any model with similar local structure. We consider the general model where the dependency relations among nodes are represented by variable model parameters (Fig. 8.1b), and then train proposals parametrized by neural networks on models with randomly generated parameters (Fig. 8.1c). The trained proposal thus work on anywhere the structure is found (Fig. 8.1d). With proposals trained for many common *motifs*, we can automatically speed up inference on unseen models.

mix effectively, the ability to escape local modes yields higher-quality solutions than the standard Gibbs sampling approach.

Related Work

There has been great interest in using learned, feedforward inference networks to generate approximate posteriors. Variational autoencoders (VAE) train an inference network jointly with the parameters of the forward model to maximize a variational lower bound [Kingma and Welling, 2013, Burda et al., 2015, Gu et al., 2015]. However, the use of a parametric variational distribution means they typically have limited capacity to represent complex, potentially multimodal posteriors, such as those incorporating discrete variables or structural uncertainty.

A related line of work has developed data-driven proposals for importance samplers [Paige and Wood, 2016, Le et al., 2017, Ritchie et al., 2016], training an inference network from prior samples which is then used as a proposal given observed evidence. In particular, [Le et al., 2017] generalize the framework to probabilistic programming, and is able to automatically generate and train a neural proposal network given an arbitrary model described in a probabilistic program. Our approach differs in that we focus on MCMC inference, allowing modular proposals for subsets of model variables that may depend on latent quantities, and exploit recurring structural motifs to generalize to new models with no additional training.

Several approaches have been proposed for adaptive block sampling, in which sets of variables exhibiting strong correlations are identified dynamically during inference, so that costly joint sampling is used only for blocks where it is likely to be beneficial [Venugopal and Gogate, 2013, Turek et al., 2016]. This is largely complementary to our current approach, which assumes the set of blocks (structural motifs) is given and attempts to learn fast approximate proposals.

Perhaps most related to our approach is recent work that trains model-specific MCMC proposals with machine learning techniques. In [Song et al., 2017a], adversarial training directly optimizes the similarity between posterior values and proposed values from a symmetric MCMC proposal. Stochastic inverses of graphical models [Stuhlmüller et al., 2013] train density estimators to speed up inference. However, both approaches have limitations on applicable models and require model-specific training using global information (samples containing all variables). Our approach is simpler and more scalable, requiring only local information and generating local proposals that can be reused both within and across different models.

At a high level, our approach of learning an approximate local update scheme can be seen as related to approximate message passing [Ross et al., 2011b, Heess et al., 2013] and learning to optimize continuous objectives [Andrychowicz et al., 2016, Li and Malik, 2017].

8.2 Meta-Learning MCMC Proposals

We propose a meta-learning approach, using a neural network to approximate the Gibbs proposal for a recurring structural motif in graphical models, and to speed up inference on unseen models without extra tuning. Crucially our proposals do *not* fix the model

parameters, which are instead provided as network input. After training with random model parametrizations, the same trained proposal can be reused to perform inference on novel models with parametrizations not previously observed.

Our inference networks are parametrized as mixture density networks [Bishop, 1994], and trained to minimize the Kullback-Leibler (KL) divergence between the true posterior conditional and the proposal by sampling instantiations of the motif. The proposals are then accepted or rejected following the Metropolis-Hastings (MH) rule [Andrieu et al., 2003], so we maintain the correct stationary distribution even though the proposals are approximate. The following sections describe our work in greater depth.

Background

Although our approach applies to arbitrary probabilistic programs, for simplicity we focus on models represented as factor graphs. A model consists of a set of variables V as the nodes of a graph $G = (V, E)$, along with a set of factors specifying a joint probability distribution $p_\Psi(V)$ described by parameters Ψ. In particular, this chapter focuses primarily on directed models, in which the factors Ψ specify the conditional probability distributions of each variable given its parents. In undirected models, such as the Conditional Random Fields (CRFs) in Sec. 8.3, the factors are arbitrary functions associated with cliques in the graph [Koller and Friedman, 2009].

Given a set of observed evidence variables, inference attempts to sample from the conditional distribution on the remaining variables. In order to construct good MCMC proposals that generalize well across a variety of inference tasks, we take the advantage of recurring *structural motifs* in graphical models, such as grids, rings, and chains [Kemp and Tenenbaum, 2008].

In this work, our goal is to train a neural network as an efficient expert proposal for a structural motif, with its inputs containing the local parameters, so that the trained proposal can be applied to different models. Within a motif, the variables are divided into a proposed set of variables that will be resampled, and a conditioning set corresponding to an approximate Markov blanket. The proposal network essentially maps the values of conditional variables and local parameters to a distribution over the proposed variables.

MCMC Proposals on Structural Motifs in Graphical Models

We associate each learned proposal with a *structural motif* that determines the shape of the network inputs and outputs. In general, structural motifs can be arbitrary subgraphs, but we are more interested in motifs that represent interesting conditional structure between two sets of variables, the block proposed variables B and the conditioning variables C. A given motif can have multiple instantiations with a model, or even across models. As a concrete example, Fig. 8.2 shows two instantiations of a structural motif of six consecutive variables in a chain model. In each instantiation, we want to approximate the conditional distribution of two middle variables given neighboring four.

(a) One instantiation. (b) Another instantiation.

Figure 8.2: Two instantiations of a structural motif in a directed chain of length seven. The motif consists of two consecutive variables and their Markov blanket of four neighboring variables. Each instantiation is separated into block proposed variables B_i (white) and conditioning variables C_i (shaded).

Definition 1. *A structural motif (B, C) (or motif in short) is an (abstract) graph with nodes partitioned into two sets, B and C, and a parametrized joint distribution $p(B, C)$ whose factorization is consistent with the graph structure. This specifies the functional form of the conditional $p(B|C)$, but not the specific parameters.*

A motif usually have many instantiations across many different graphical models.

Definition 2. *For a graphical model $(G = (V, E), \Psi)$, an* instantiation *(B_i, C_i, Ψ_i) of a motif (B, C) includes*

1. *a subset of the model variables $(B_i, C_i) \subseteq V$ such that the induced subgraph on (B_i, C_i) is isomorphic to the motif (B, C) with the partition preserved by the isomorphism (so nodes in B are mapped to B_i, and C to C_i), and*

2. *a subset of model parameters $\Psi_i \subseteq \Psi$ required to specify the conditional distribution $p_{\Psi_i}(B|C)$.*

We would typically define a structural motif by first picking out a block of variables B to jointly sample, and then selecting a conditioning set C. Intuitively, the natural choice for a conditioning set is the Markov blanket, $C = \mathrm{MB}(B)$. However, this is not a fixed requirement, and C could be either a subset or superset of it (or neither). We might deliberately choose to use some alternate conditioning set C, e.g. , a subset of the Markov blanket to gain a more computationally efficient proposal (with a smaller proposal network), or a superset with the idea of learning longer-range structure. More fundamentally, however, Markov blankets depend on the larger graph structure might not be consistent across instantiations of a given motif (e.g. , if one instantiation has additional edges connecting B_i to other model variables not in C_i). Allowing C to represent a generic conditioning set leaves us with greater flexibility in instantiating motifs.

Formally, our goal is to learn a Gibbs-like block proposal $q(B_i|C_i; \Psi_i)$ for all possible instantiations (B_i, C_i, Ψ_i) of a structural motif that is close to the true conditional in the sense that

$$\forall (B_i, C_i, \Psi_i),\ \forall c_i \in \mathrm{supp}(C_i), q(B_i; c_i, \Psi_i) \approx p_{\Psi_i}(B_i|C_i = c_i). \tag{8.1}$$

This provides another view of this approximation problem. If we choose the motif to have complex structures in each instantiation, the conditionals $p_{\Psi_i}(B_i|C_i = c_i)$ can often be quite

different for different instantiations, and thus difficult to approximate. Therefore, choosing what is a structural motif represents a trade-off between generality of the proposal and easiness to approximate. While our approach works for any structural motif complying with the above definition, we suggest using common structures as motifs, such as chain of certain length as in Fig. 8.2. In principle, recurring motifs could be automatically detected, but in this work, we focus on hand-identified common structures.

Parametrizing Neural Block Proposals

We choose mixture density networks (MDN) [Bishop, 1994] as our proposal network parametrization. An MDN is a form of neural network whose outputs parametrize a mixture distribution, where in each mixture component the variables are uncorrelated.

In our case, a neural block proposal is a function q_θ parametrized by a MDN with weights θ. The function q_θ represents proposals for a structural motif (B, C) by taking in current values of C_i and local parameters Ψ_i, and outputting a distribution over B_i. The goal is to optimize θ so that q_θ is close to the true conditional.

In the network output, mixture weights are represented explicitly. Within each mixture component, distributions of bounded discrete variables are directly represented as independent categorical probabilities, and distributions of continuous variables are represented as isotropic Gaussians with mean and variance. To avoid degenerate proposals, we threshold the variance of each Gaussian component to be at least 10^{-5}.

Training Neural Block Proposals

Loss function for a specific instantiation: Given a particular motif instantiation, we use the KL divergence $D(p_{\Psi_i}(B_i|C_i) \parallel q_\theta(B_i; C_i, \Psi_i))$ as the measure of closeness between our proposal and the true conditional in Eq. 8.1. Taking into account all possible values $c_i \in \text{supp}(C_i)$, we consider the expected divergence over C_i's prior:

$$\mathbb{E}_{C_i}[D(p_{\Psi_i}(B_i|C_i) \parallel q_\theta(B_i; C_i, \Psi_i))] = -\mathbb{E}_{B_i,C_i}[\log q_\theta(B_i; C_i, \Psi_i)] + \text{constant}. \tag{8.2}$$

The second term is independent of θ. So we define the loss function on (B_i, C_i, Ψ_i) as

$$\tilde{L}(\theta; B_i, C_i, \Psi_i) = -\mathbb{E}_{B_i,C_i}[\log q_\theta(B_i; C_i, \Psi_i)].$$

Meta-training over many instantiations: To train a generalizable neural block proposal, we generate a set of random instantiations and optimize the loss function over all of them. Assuming a distribution over instantiations $\mathcal{P}$, our goal is to minimize the overall loss

$$L(\theta) = \mathbb{E}_{(B_i,C_i,\Psi_i)\sim\mathcal{P}}[\tilde{L}(\theta; B_i, C_i, \Psi_i)] = -\mathbb{E}_{(B_i,C_i,\Psi_i)\sim\mathcal{P}}[\mathbb{E}_{B_i,C_i}[\log q_\theta(B_i; C_i, \Psi_i)]], \tag{8.3}$$

which is optimized with minibatch SGD in our experiments.

There are different ways to design the motif instantiation distribution $\mathcal{P}$. One approach is to find a distribution over model parameter space, and attach the random parametrizations

Algorithm 3: Neural Block Sampling

Input: Graphical model (G, Ψ), observations y, motifs $\{(B^{(m)}, C^{(m)})\}_m$, and their instantiations $\{(B_i^{(m)}, C_i^{(m)}, \Psi_i^{(m)})\}_{i,m}$ detected in (G, Ψ).

for each motif $B^{(m)}, C^{(m)}$ **do**
  if proposal trained for this motif exists **then**
    $q^{(m)} \longleftarrow$ trained neural block proposal
  else
    Train neural block proposal $q_\theta^{(m)}$ using SGD by Eq. 8.3 on its instantiations $\{(B_i^{(m)}, C_i^{(m)}, \Psi_i^{(m)})\}_i$
  end if
end for
$x \longleftarrow$ initialize state
for timestep **in** $1 \dots T$ **do**
  Propose $x' \leftarrow$ proposal $q_\theta^{(m)}$ on some instantiation $(B_i^{(m)}, C_i^{(m)}, \Psi_i^{(m)})$
  Accept or reject according to MH rule
end for
return MCMC samples

Ψ_i to (B_i, C_i). Practically, it is also viable to find a training dataset of models that contains a large number of instantiations. Both approaches are discussed in detail and experimented in the experiment section.

Neural block sampling: The overall MCMC sampling procedure with meta-proposals is outlined in Algorithm 3, which supports building a library of neural block proposals trained on common motifs to speed up inference on previously unseen models.

8.3 Experiments

In this section, we evaluate our method of learning neural block proposals against single-site Gibbs sampler as well as several model-specific MCMC methods. We focus on three most common structural motifs: grids, mixtures and chains. In all experiments, we use the following guideline to design the proposal: (1) using small underlying MDNs (we pick networks with two hidden layers and elu activation [Clevert et al., 2015]), and (2) choosing an appropriate distribution to generate parameters of the motif such that the generated parameters could cover the whole space as much as possible. More experiments details and an additional experiment are available in the supplementary materials.

Grid Models

We start with a common structural motif in graphical models, grids. In this section, we focus on binary-valued grid models of all sorts for their relative easiness to directly compute

posteriors. To evaluate MCMC algorithms, we compare the estimated posterior marginals $\hat{P}$ against true posterior marginals P computed using IJGP [Mateescu et al., 2010]. For each inference task with N variables, we calculated the error $\frac{1}{N}\sum_{i=1}^{N}\left|\hat{P}(X_i=1)-P(X_i=1)\right|$ as the mean absolute deviation of marginal probabilities.

General Binary-Valued Grid Models

We consider the motif in Fig. 8.3, which is instantiated in every binary-valued grid Bayesian networks (BN). Our proposal takes in the conditional probability tables (CPTs) of all 23 variables as well as the current values of 14 conditioning variables, and outputs a distribution over the 9 proposed variables.

To sample over all possible binary-valued grid instantiations, we generate random grids by sampling each CPT entry i.i.d. from a mixed distribution of this following form:

$$\begin{cases} [0,1] & \text{w.p. } \frac{p_{\text{determ}}}{2} \\ [1,0] & \text{w.p. } \frac{p_{\text{determ}}}{2} \\ \text{Dirichlet}(\boldsymbol{\alpha}) & \text{w.p. } 1-p_{\text{determ}}, \end{cases} \tag{8.4}$$

where $p_{\text{determ}} \in [0,1]$ is the probability of the CPT entry being deterministic. Our proposal is trained with $p_{\text{determ}} = 0.05$ and $\boldsymbol{\alpha} = (0.5, 0.5)$.

To test the generalizability of our trained proposal, we generate random binary grid instantiations using distributions with various p_{determ} and $\boldsymbol{\alpha}$ values, and compute the KL divergences between the true conditionals and our proposal outputs on 1000 sampled instantiations from each distribution. Fig. 8.5 shows the histograms of divergence values from 4 very different distributions, including the one used for training (top left). The resulting histograms show mostly small divergence values, and are nearly indistinguishable, even though one distribution has $p_{\text{determ}} = 0.8$ and the proposal is only trained with $p_{\text{determ}} = 0.05$. This shows that our approach is able to generally and accurately approximate true conditionals, despite only being trained with an arbitrary distribution.

We evaluate the performance of the trained neural block proposal on all 180 grid BNs up to 500 nodes from UAI 2008 inference competition. In each epoch, for each latent variable, we try to identify and propose the block as in Fig. 8.3 with the variable located at center. If this is not possible, e.g. , the variable is at boundaries or close to evidence, single-site Gibbs resampling is used instead.

Fig. 8.6 shows the performance of both our method and single-site Gibbs in terms of error integrated over time for all 180 models. The models are divided into three classes, grid-50, grid-75 and grid-90, according to the percentage of deterministic relations. Our neural block sampler significantly outperforms Gibbs sampler in nearly every model. We notice that the improvement is less significant as the percentage of deterministic relations increases. This is largely due to that the above proposal structure in Fig. 8.3 can only easily handle dependency among the 9 proposed nodes. We expect an increased block size to yield stronger performance on models with many deterministic relations.

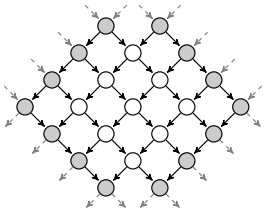


Figure 8.3: Motif for general grid models. Conditioning variables (shaded) form the Markov blanket of proposed variables (white). Dashed gray arrows show possible but irrelevant dependencies.

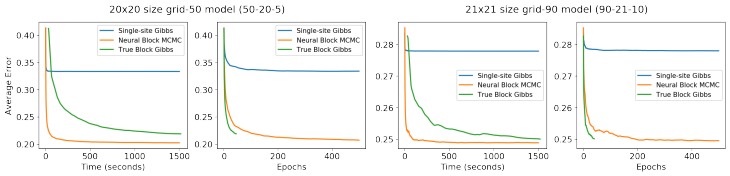


Figure 8.4: Sample runs comparing single-site Gibbs, Neural Block Sampling, and block Gibbs with true conditionals. For each model, we compute 10 random initializations and run three algorithms for 1500s on each. Epochs plots are cut off at 500 epochs to better show the comparison because true block Gibbs finishes far less epochs within given time. `50-20-5` and `90-21-10` are identifiers of these two models in the competition.

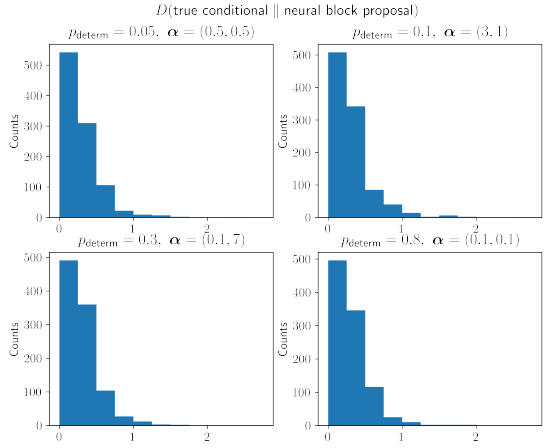


Figure 8.5: KL divergences between the true conditionals and our proposal outputs on 1000 sampled instantiations from 4 distributions with different p_{determ} and $\boldsymbol{\alpha}$. Top left is the distribution used in training. Our trained proposal is able to generalize on arbitrary binary grid models.

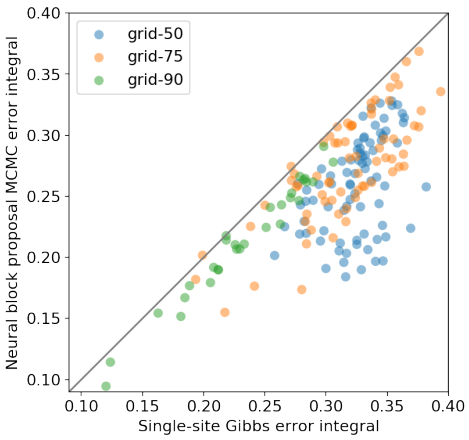


Figure 8.6: Performance comparison on 180 grid models from UAI 2008 inference competition. Each mark represents error integrals for both single-site Gibbs and our method in a single run over 1200s inference.

Furthermore, we compare our proposal against single-site Gibbs, and exact block Gibbs with identical proposal block, on grid models with different percentages of deterministic relations in Fig. 8.4. Single-site Gibbs performs worst on both models due to quickly getting stuck in local modes. Between the two block proposals, neural block sampling performs better in error w.r.t. time due to shorter computational time. However, because the neural block proposal is only an approximate of the true block Gibbs proposal, it is worse in terms of error w.r.t. epochs, as expected. Detailed comparisons on more models are available in the

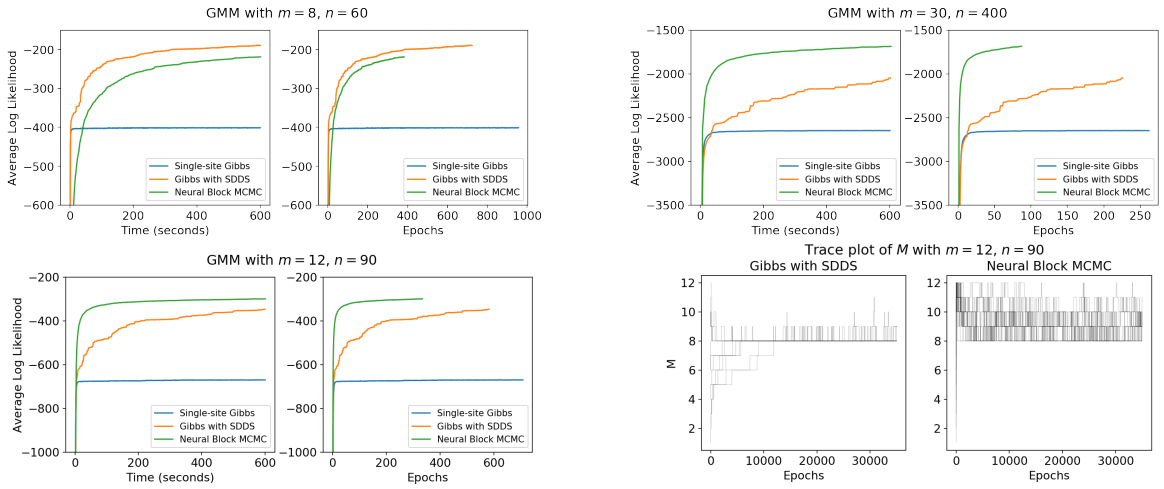


Figure 8.7: **All except bottom right:** Average log likelihoods of MCMC runs over 200 tasks for total 600s in various GMMs. **Bottom right:** Trace plots of M over 12 runs from initialization with different M values on a GMM with $m = 12$, $n = 90$. Our approach explores sample space much faster than Gibbs with SDDS.

supplementary material.

Additionally, our approach can be used model-specifically by training only on instantiations within a particular model. In supplementary materials, we demonstrate that our method achieves comparable performance with a more advanced task-specific MCMC method, Inverse MCMC [Stuhlmüller et al., 2013].

Gaussian Mixture Model with Unknown Number of Components

We next consider open-universe Gaussian mixture models (GMMs), in which the number of mixture components is unknown, subject to a prior. Similarly to Dirichlet process GMMs, these are typically treated with hand-designed model-specific split-merge MCMC algorithms.

Consider the following GMM. n points $\mathbf{x} = \{x_i\}_{i=1,\ldots,n}$ are observed, and come uniformly randomly from one of M (unknown) active mixtures, with $M \sim \text{Unif}\{1, 2, \ldots, m\}$. Our task is to infer the posterior of mixture means $\boldsymbol{\mu} = \{\mu_j\}_{j=1,\ldots,M}$, their activity indicators $\mathbf{v} = \{v_j\}_{j=1,\ldots,M}$, and the labels $\mathbf{z} = \{z_i\}_{i=1,\ldots,n}$, where z_i is the mixture index x_i comes from. Since M is determined by $\mathbf{v}$, in this experiment, we always calculate $M = \sum_j v_j$ instead of sampling M.

Such GMMs have many nearly-deterministic relations, e.g. , $p(v_j = 0, z_i = j) = 0$, causing vanilla single-site Gibbs failing to jump across different M values. Split-merge MCMC algorithms, e.g. , Restricted Gibbs split-merge (RGSM) [Jain and Neal, 2004] and Smart-Dumb/Dumb-Smart (SDDS) [Wang and Russell, 2015], use hand-designed MCMC moves to solve such issues. In our framework, it's possible to deal with such relations with a proposal block including all of $\mathbf{z}$, $\boldsymbol{\mu}$ and $\mathbf{v}$. However, doing so requires significant training and

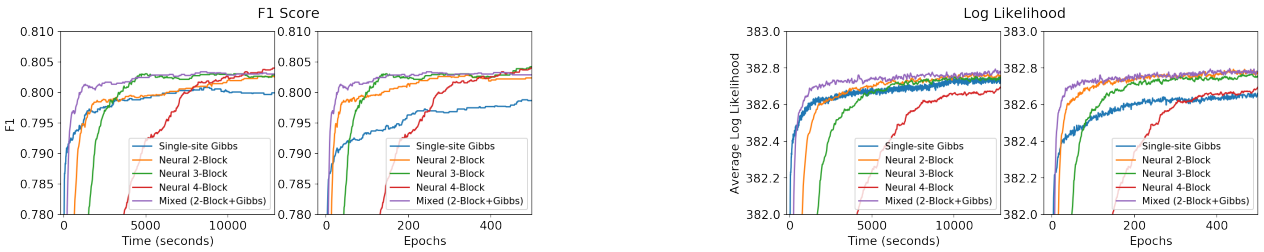


Figure 8.8: Average F1 scores and average log likelihoods over entire test dataset. In each epoch, all variables in every test MRF is proposed roughly once for all algorithms. F1 scores are measured using states with highest likelihood seen over Markov chain traces. To better show comparison, epoch plots are cut off at 500 epochs and time plots at 12850s. Log likelihoods shown don't include normalization constant.

inference time (due to larger proposal network and larger proposal block), and the resulting proposal can not generalize to GMMs of different sizes.

In order to apply the trained proposal to differently sized GMMs, we choose the motif to propose q_θ for two arbitrary mixtures (μ_i, v_i) and (μ_j, v_j) conditioned on all other variables *excluding* $\mathbf{z}$, and instead consider the model with $\mathbf{z}$ variables collapsed. The inference task is then equivalent to first sampling $\boldsymbol{\mu}, \mathbf{v}$ from the collapsed model $p(\boldsymbol{\mu}, \mathbf{v}|\mathbf{x})$, and then $\mathbf{z}$ from $p(\mathbf{z}|\boldsymbol{\mu}, \mathbf{v}, \mathbf{x})$. We modify the algorithm such that the proposal from q_θ is accepted or rejected by the MH rule on the *collapsed* model. Then $\mathbf{z}$ is resampled from $p(\mathbf{z}|\boldsymbol{\mu}, \mathbf{v}, \mathbf{x})$. This approach is less sensitive to different n values and performs well in variously sized GMMs. More details are available in the supplementary material.

We train with a small GMM with $m = 8$ and $n = 60$ as the motif, and apply the trained proposal on GMMs with larger m and n by randomly selecting 8 mixtures and 60 points for each proposal. Fig. 8.7 shows how the our sampler performs on GMM of various sizes, compared against split-merge Gibbs with SDDS. We notice that as model gets larger, Gibbs with SDDS mixes more slowly, while neural block sampling still mixes fairly fast and outperforms Gibbs with SDDS. Bottom right of Fig. 8.7 shows the trace plots of M for both algorithms over multiple runs on the same observations. Gibbs with SDDS takes a long time to find a high likelihood explanation and fails to explore other possible ones efficiently. Our proposal, on the other hand, mixes quickly among the possible explanations.

Named Entity Recognition (NER) Tagging

Named entity recognition (NER) is the task of inferring named entity tags for words in natural language sentences. One way to tackle NER is to train a conditional random field (CRF) model representing the joint distribution of tags and word features [Liang et al., 2008]. In test time, we use the CRF build a chain Markov random field (MRF) containing only tags variables, and apply MCMC methods to sample the NER tags. We use a dataset of 17494

sentences from CoNLL-2003 Shared Task[1]. The CRF model is trained with AdaGrad [Duchi et al., 2011] through 10 sweeps over the training dataset.

Our goal is to train good neural block proposals for the chain MRFs built for test sentences. Experimenting with different chain lengths, we train three proposals, each for a motif of two, three, or four consecutive proposed tag variables and their Markov blanket. These proposals are trained on instantiations within MRFs built from the training dataset for the CRF model.

We then evaluate the learned neural block proposals on the previously unseen test dataset of 3453 sentences. Fig. 8.8 plots the performance of neural block sampling and single-site Gibbs w.r.t. both time and epochs on the entire test dataset. As block size grows larger, learned proposal takes more time to mix. But eventually, block proposals generally achieve better performance than single-site Gibbs in terms of both F1 scores and log likelihoods. Therefore, as shown in the figure, a mixed proposal of single-site Gibbs and neural block proposals can achieve better mixing without slowing down much. As an interesting observation, neural block sampling sometimes achieves higher F1 scores even before surpassing single-site Gibbs in log likelihood, implying that log likelihood is at best an imperfect proxy for performance on this task.

8.4 Additional Details

Experiment Details

As mentioned in Sec. 8.3, parametrizing MDNs in all experiments have elu activation and two hidden layers each of size $\lambda \max\{\text{input size}, \text{output size}\}$, where $4 \leq \lambda \leq 5$ depending on the task, and output the proposal distribution as a mixture of $4 \leq m \leq 16$ components.

General Binary-Valued Grid Models

For directed binary-valued grid models, we use higher amount of MDN mixtures than other experiments because more variables are proposed and general discrete BNs can have highly multi-modal conditionals.

Architecture The underlying MDN has 106-480-480-120 network structure, mapping the CPTs of all 23 motif variables and 14 conditioning variable values to the proposal distribution of 9 proposed variables as a mixture of 12 components.

Additional Sample Runs We provide additional sample runs on four grid models with various percentages of deterministic relations in Fig. 8.9.

[1] https://www.clips.uantwerpen.be/conll2003/ner/

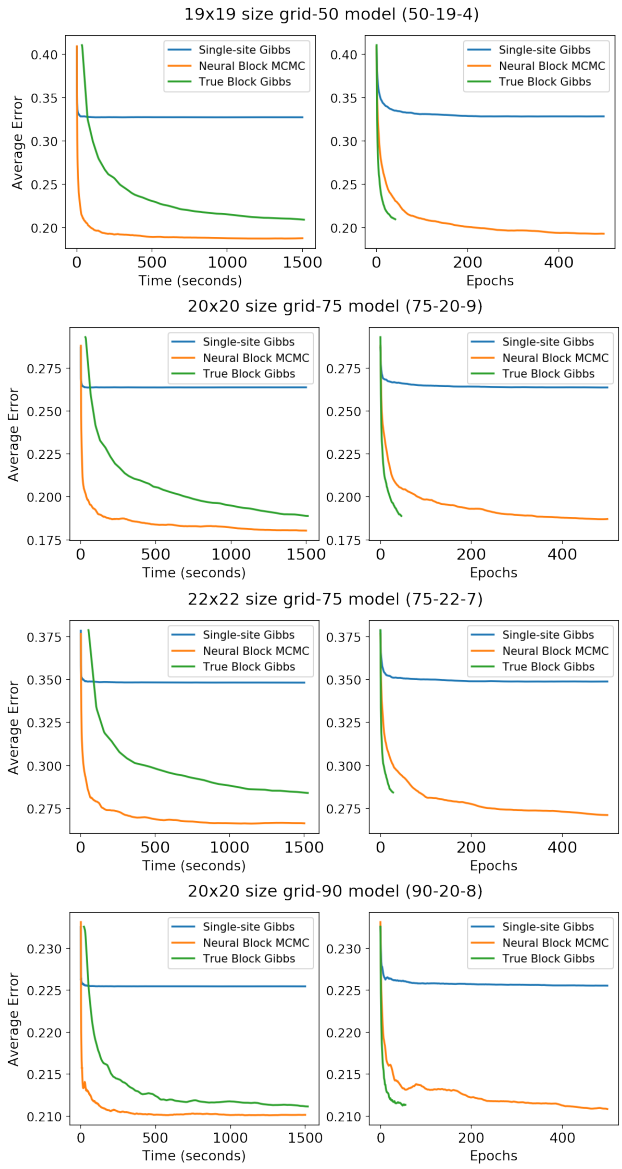


Figure 8.9: Additional sample runs with single-site Gibbs, neural block MCMC, and block Gibbs with true conditionals on UAI 2008 grid models. These results are obtained in same setting as Fig. 8.4 of main paper. For each model, we compute 10 random initializations and run three algorithms for 1500s on each one. Plots show average error for each algorithm. Epochs plots are cut off at 500 epochs to better show the comparison. "grid-k" represents that the model has k% deterministic relations.

Gaussian Mixture Model with Unknown Number of Components

Formal Model Formally, the model can be written as

$$\begin{aligned} M &\sim \text{Unif}\{1, 2, \ldots, m\} \\ \mu_j &\sim \mathcal{N}(0, \sigma_\mu^2 I) && j = 1, \ldots, m \\ \mathbf{v}|M &\sim \text{Unif}\{a \in \{0,1\}^m : \sum_j a_j = M\} \\ z_i|\mathbf{v} &\sim \text{Unif}\{j : v_j = 1\} && i = 1, \ldots, n \\ x_i|z_i, \boldsymbol{\mu} &\sim \mathcal{N}(\mu_{z_i}, \sigma^2 I) && i = 1, \ldots, n, \end{aligned}$$

where m and n are model parameters, and $\sigma_\mu^2 = 4$ and $\sigma^2 = 0.1$ are fixed constants.

Architecture Our neural block proposal is trained using the GMM with $m = 8$ max mixtures and $n = 60$ data points. It proposes two mixture components (μ_j, v_j)s through underlying MDN of 156-624-624-36 network structure. The MDN's inputs include 60 observed points $\mathbf{x} = \{x_i\}_i$, 8 component means $\boldsymbol{\mu} = \{\mu_j\}_j$ and component active indicators $\mathbf{v} = \{v_j\}_j$ with values for the two proposed mixtures replaced by zeros. Orders for $\mathbf{x}$, $\boldsymbol{\mu}$ and $\mathbf{v}$ are such that they are sorted along the first principle component of $\mathbf{x}$ to break symmetry. In addition, the inputs also contain the principle component and the indicators of which components are being proposed. The MDN outputs a proposal distribution over the two (μ_j, v_j)s as a mixture of 4 MDN components.

Breaking the Symmetry In training, such mixture models have symmetries that must be broken before being used as input to the neural network [Nishihara et al., 2013]. In particular, the mixtures $\{(v_j, \mu_j)\}_j$ can be permuted in $m!$ ways and the points $\{(z_i, x_i)\}_i$ in $n!$ ways. Following a similar procedure by [Le et al., 2017], we sort these values according the first principal component of $\mathbf{x}$, and also feed the first principal component vector into the MDN.

Proposal and Inference with z Collapsed In order to avoid nearly-deterministic relations, e.g. , $p(v_j = 0, z_i = j) = 0$, and still train a general proposal unconstrained by n, we choose to consider the collapsed model without $\mathbf{z}$. We first experiment on the intuitive approach which adds a resampling step for $\mathbf{z}$ in the proposal. At each proposal step, trained proposal q_θ is first used to propose new mixtures $\boldsymbol{\mu}'$ and $\mathbf{v}'$, and then $\mathbf{z}$ is proposed from $p(\mathbf{z}|\boldsymbol{\mu}', \mathbf{v}', \mathbf{x})$. The MH rule is applied lastly to either accept or reject all proposed values. While this method gives good performance in small models, it suffers greatly from low acceptance ratio as n, the number of observed points $\mathbf{z}$, grows large. Therefore, we eventually choose the approach described in Sec. 8.3, i.e. , applying the MH rule on the *collapsed* model with $\mathbf{z}$ resampled from $p(\mathbf{z}|\boldsymbol{\mu}, \mathbf{v}, \mathbf{x})$ afterwards. Since the acceptance ratio no longer depends on n, this approach behaves much more scalable than the first one in our experiments. It outperforms SDDS split-merge MCMC in GMMs of various sizes, as shown in Fig. 8.7 of main paper.

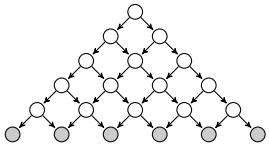


Figure 8.10: Small version of the triangle grid model in experiment Sec. 8.4. Evidence nodes (shaded) are at bottom layer. The actual network has 15 layers and 120 nodes.

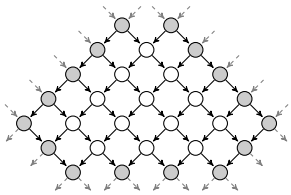


Figure 8.11: Motif for the model in Fig. 8.10. Conditioning variables (shaded) form the Markov blanket of proposed variables (white). Dashed gray arrows are possible irrelevant dependencies.

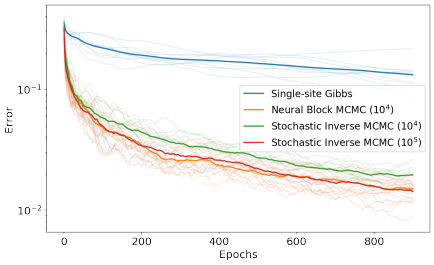


Figure 8.12: Error w.r.t. epochs on the triangle model in Fig. 8.10. Semi-transparent lines show individual MCMC runs. Opaque lines show average error over 10 MCMC runs for each algorithm. Numbers in parentheses are the amounts of training data.

NER Tagging

Architecture The underlying MDN has two hidden layers each of size $4 \times \max\{\text{input size}, \text{output size}\}$, with output size varying according to the number of proposed variables. It maps local CRF parameters of all motif variables and conditioning variable values to the NER tag proposal for consecutive proposed variables as a mixture of 4 components.

Additional Experiment

Comparison with Inverse MCMC

Neural block proposals can also be used model-specifically by training only on instantiations within a particular model. In this subsection, we demonstrate that our method achieves comparable performance with a more complex task-specific MCMC method, Inverse MCMC [Stuhlmüller et al., 2013].

Figure 8.10 illustrates the triangle grid network used in this experiment, which is identical to what [Stuhlmüller et al., 2013] used to evaluate Inverse MCMC. For our method, we chose the motif shown in Fig. 8.11. The proposal is trained on all instantiations in this triangle model.

Inverse MCMC is an algorithm that builds auxiliary data structures offline to speed up inference. Given an inference task, it trains an inverse graph for each latent variable where the latent variable is at bottom and evidence variables are at top. These graphs are then used as MCMC proposals.

It is difficult to compare these two methods w.r.t. time. While both methods require offline training, Inverse MCMC needs to train from scratch if the set of evidence nodes changes, yet neural block sampling only needs one-time training for different inference tasks

on this model. In this experiment, for each inference epoch, both methods propose about 10.5 values on average per latent variable. Figure 8.12 shows a more meaningful comparison of error w.r.t. epochs. Our learned neural block proposal, trained using 10^4 samples, achieves comparable performance with Inverse MCMC, which is trained using 10^5 samples and builds model-specific data structures (inverse graphs).

Architecture The underlying MDN has 161-1120-1120-224 network structure, mapping the CPTs of all 29 motif variables and 16 conditioning variable values to the proposal distribution of 13 proposed variables as a mixture of 16 components.

Inverse MCMC Setting In this experiment, we run Inverse MCMC with frequency density estimator trained with posterior samples, proposal block size up to 20 and Gibbs proposals precomputed, following the original approach of [Stuhlmüller et al., 2013].

8.5 Summary

This chapter explores the idea of meta-learning generalizable approximate block Gibbs proposals and applies it to an challenging NLP task, named entity recognition. Our meta-proposals are trained offline and can be applied directly to novel models given only a common set of structural motifs. Experiments show that the neural block sampling approach outperforms standard single-site Gibbs in both convergence speed and sample quality and achieve comparable performance against model-specialized methods. In will be an interesting system design problem to investigate, when given a library of trained block proposals, how an inference system in a probabilistic programming language can automatically detect the common structural motifs and (adaptively) apply appropriate samplers to help convergence for more general real-world applications.

Additionally, from the meta-learning perspective, our method is based on meta-training, i.e. , training over a variety of motif instantiations. At test time, the learned proposal does not adapt to new scenarios after meta-training. While in many meta-learning works in reinforcement learning [Finn et al., 2017a, Duan et al., 2016b], a meta-trained agent can further adapt the learned policy to unseen environments via a few learning steps under the assumption that a reward signal is accessible at test time. In our setting, we can similarly adopt such fast adaptation scheme at test time to further improve the quality of proposed samples by treating the acceptance rate as a test time reward signal. We leave this as a future work.

Chapter 9

Conclusion

This thesis studies a fundamental challenge in Artificial Intelligence, i.e., how to build learning agents that can generalize from training scenarios to unseen test cases, which is the key prerequisite to bring any AI application to our daily life. Chapter 1 introduces a mathematical formulation of this generalization problem (Eq. 1.1, 1.2) and further proposes 4 solution-design principles focusing on 4 different aspects of the generalization challenge respectively (Sec. 1.3), namely (1) policy representation, (2) diverse training, (3) effective learning objective, and (4) improved training paradigm. The remaining chapters examine these 4 principles comprehensively by considering 4 concrete domains, i.e., discrete graph (Part I), 3D visual navigation (Part II), complex multi-agent games (Part III) and natural language processing applications (Part IV).

In Part I, Chapter 2 follows *principle (1)* by proposing a novel policy representation, value iteration network, which allows an agent to learn a generic planning computation for path finding on graph structures, including simple gridworlds and real-world webgraphs. In Part II, Chapter 3 follows *principle (2)* by developing a new large-scale environment, House3D, and analyzing the effectiveness of a variety of data augmentation techniques. Chapter 4 follows *principle (1)* by designing a memory architecture, Bayesian Relational Memory, for visual semantic planning. In Part III, Chapter 5 firstly studies the intrinsic challenge in multi-agent games due to simultaneous learning agents and then introduces an effective and general learning algorithm for multi-agent games, MADDPG, according to *principle (4)*. Chapter 6 follows *principle (3)* by introducing the minimax concept into the learning objective of the MADDPG algorithm. In Part IV, Chapter 7 considers the relation extraction task and improves the robustness of learned models via an adversarial training objective under *principle (3)*. Chapter 8 considers the named entity recognition task and proposes a novel meta-learning paradigm to effectively improve the inference performance of a learned model on testing texts under *principle (4)*.

The works presented in this thesis empirically demonstrate that by designing algorithmic techniques under the proposed principles, we can improve the generalization ability of learning agents in a wide range of challenging domains. In addition, this thesis also suggests some open questions for future research. Some of the most critical questions from my perspective

are described below.

(1) *What is the necessary inductive bias in the learning paradigm for generalization?*

In classical AI, which typically utilizes model-based approaches over symbolic abstractions, generalization is naturally guaranteed. For example, any standard search algorithm can successfully solve any gridworld navigation task while it takes efforts for deep learning approaches to perform reasonably well (see Chapter 2). It is a recent trend in the community to utilize concepts from classical AI, such as objects [Kulkarni et al., 2016, Devin et al., 2018], relations [Zambaldi et al., 2019] and planning [Kansky et al., 2017, Finn and Levine, 2017], when building deep learning agents. In this thesis, Chapter 2 and Chapter 4 also inherit similar ideas. However, since most real-world problems are high-dimensional, it remains non-trivial and often requires strong domain priors to extract meaningful low-dimensional abstractions in general. Moreover, some recent evidences, such as MuZero from DeepMind [Schrittwieser et al., 2019], show that with a large amount of training data, neural agents can automatically learn specialized latent representations without any *explicit* task prior introduced at training and outperform existing methods with strong domain knowledge. So, which direction should we preferred? Explicitly introducing more human insights or simply forcing the agents to discover its own representations from data and tasks? It is not yet clear which specific inductive bias would become the most necessary and the most critical ingredient for building intelligent agents.

(2) *What is the limit of success through diverse training?*

This question follows the previous one. Some recent works, including Chapter 3, show that by simply combining more training data and massive compute power, we can significantly improve the generalization performances of learning agents without big algorithmic changes. For example, visual models trained with more images have substantially higher recognition accuracy [Mahajan et al., 2018]; language models pretrained with massive corpus can generate extremely fluent English paragraphs [Devlin et al., 2019, Radford et al., 2019]; deep RL agents trained via self-play and massive compute can defeat professional players in Dota 2 [OpenAI et al., 2019b] and StarCraft II [Vinyals et al., 2019]; by performing domain randomization in the simulator, a trained dexterous hand can solve a Rubik's cube in the real world [OpenAI et al., 2019a]. To some extent, these empirical results seemingly indicate that AGI could be potentially achieved by collecting more data and training larger models — although most of use are not convinced by such simple roadmap. So, what is the limit of intelligence we can achieve through massive data and compute? Which problems are fundamentally hard while which problems can be simply solved by simply increasing the amount of compute? This question also suggests that the compute should be another important factor of inductive bias to consider for building generalizable agents.

(3) *How can we efficiently search for effective diverse policies?*

This challenge is particularly severe for multi-agent scenarios (Part III): when the number of agent increases, the problem complexity grows exponentially and therefore it is considerably harder to discover good policies. Notably, due to the fact that self-play is the most widely used training paradigm for multi-agent games, how strong a learned agent is directly depends on the quality of its training partners. Hence, how to efficiently and effectively discover *diverse* (counter-)strategies during self-play training is a necessity for building any agents that can generalize in multi-agent scenarios.

At the moment, there are no certain answers to these questions, but I hope this thesis can provide useful insights to the community and serve as a solid step towards our ultimate goal of AGI, with my optimism.

* * *

"Our Conquest is the Sea of Stars!"

Bibliography

[Abadi and Andersen, 2016] Abadi, M. and Andersen, D. G. (2016). Learning to protect communications with adversarial neural cryptography. *CoRR*, abs/1610.06918.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283. USENIX Association.

[Adel et al., 2016] Adel, H., Roth, B., and Schütze, H. (2016). Comparing convolutional neural networks to traditional models for slot filling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 828–838.

[Al-Rfou et al., 2016] Al-Rfou, R., Alain, G., Almahairi, A., Angermüller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Snyder, J. B., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P. L., Cho, K., Chorowski, J., Christiano, P. F., Cooijmans, T., Côté, M., Côté, M., Courville, A. C., Dauphin, Y. N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Kahou, S. E., Erhan, D., Fan, Z., Firat, O., Germain, M., Glorot, X., Goodfellow, I. J., Graham, M., Gülçehre, Ç., Hamel, P., Harlouchet, I., Heng, J., Hidasi, B., Honari, S., Jain, A., Jean, S., Jia, K., Korobov, M., Kulkarni, V., Lamb, A., Lamblin, P., Larsen, E., Laurent, C., Lee, S., Lefrançois, S., Lemieux, S., Léonard, N., Lin, Z., Livezey, J. A., Lorenz, C., Lowin, J., Ma, Q., Manzagol, P., Mastropietro, O., McGibbon, R., Memisevic, R., van Merriënboer, B., Michalski, V., Mirza, M., Orlandi, A., Pal, C. J., Pascanu, R., Pezeshki, M., Raffel, C., Renshaw, D., Rocklin, M., Romero, A., Roth, M., Sadowski, P., Salvatier, J., Savard, F., Schlüter, J., Schulman, J., Schwartz, G., Serban, I. V., Serdyuk, D., Shabanian, S., Simon, É., Spieckermann, S., Subramanyam, S. R., Sygnowski, J., Tanguay, J., van Tulder, G., Turian, J. P., Urban, S., Vincent, P., Visin, F., de Vries, H., Warde-Farley, D., Webb, D. J., Willson, M., Xu, K., Xue, L., Yao, L., Zhang, S., and Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. *CoRR*, abs/1605.02688.

[Al-Shedivat et al., 2018] Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. (2018). Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*.

[Anand et al., 2018] Anand, A., Belilovsky, E., Kastner, K., Larochelle, H., and Courville, A. (2018). Blindfold baselines for embodied QA. *CoRR*, abs/1811.05013.

[Anderson et al., 2018a] Anderson, P., Chang, A. X., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., and Zamir, A. R. (2018a). On evaluation of embodied navigation agents. *CoRR*, abs/1807.06757.

[Anderson et al., 2018b] Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A. (2018b). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

[Andreas et al., 2017] Andreas, J., Klein, D., and Levine, S. (2017). Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning*, pages 166–175. JMLR. org.

[Andrieu et al., 2003] Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine learning*, 50(1):5–43.

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

[Armeni et al., 2016] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3D semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543.

[Beattie et al., 2016] Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. (2016). Deepmind lab. *CoRR*, abs/1612.03801.

[Bellemare et al., 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[Berg-Kirkpatrick et al., 2014] Berg-Kirkpatrick, T., Andreas, J., and Klein, D. (2014). Unsupervised transcription of piano music. In *Advances in Neural Information Processing Systems*, pages 1538–1546.

[Bertsekas, 2012] Bertsekas, D. (2012). *Dynamic Programming and Optimal Control, Vol II.* Athena Scientific, 4th edition.

[Bishop, 1994] Bishop, C. M. (1994). Mixture density networks.

[Bollacker et al., 2008] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250. ACM.

[Bonin-Font et al., 2008] Bonin-Font, F., Ortiz, A., and Oliver, G. (2008). Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263–296.

[Boutilier, 1996] Boutilier, C. (1996). Learning conventions in multiagent stochastic domains using likelihood estimates. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, pages 106–114. Morgan Kaufmann Publishers Inc.

[Brock et al., 2019] Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI gym. *CoRR*, abs/1606.01540.

[Brodeur et al., 2017] Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., Rouat, J., Larochelle, H., and Courville, A. (2017). HoME: A household multimodal environment. *CoRR*, abs/1711.11017.

[Brown and Sandholm, 2019] Brown, N. and Sandholm, T. (2019). Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890.

[Burda et al., 2015] Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *CoRR*, abs/1509.00519.

[Busoniu et al., 2008] Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, 38(2):156.

[Cai et al., 2016] Cai, R., Zhang, X., and Wang, H. (2016). Bidirectional recurrent convolutional neural network for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 756–765.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75.

[Chalkiadakis and Boutilier, 2003] Chalkiadakis, G. and Boutilier, C. (2003). Coordination in multiagent reinforcement learning: a Bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 709–716. ACM.

[Chang et al., 2017] Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., and Zhang, Y. (2017). Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision (3DV)*.

[Chaplot et al., 2018] Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. (2018). Gated-attention architectures for task-oriented language grounding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Chen et al., 2015] Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730.

[Chen et al., 2019] Chen, K., de Vicente, J. P., Sepulveda, G., Xia, F., Soto, A., Vazquez, M., and Savarese, S. (2019). A behavioral approach to visual navigation with graph localization networks. *CoRR*, abs/1903.00445.

[Chen et al., 2018] Chen, X., Li, L.-J., Fei-Fei, L., and Gupta, A. (2018). Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7239–7248.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

[Christiano et al., 2016] Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *CoRR*, abs/1610.03518.

[Ciresan et al., 2012] Ciresan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649.

[Clevert et al., 2015] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs). *CoRR*, abs/1511.07289.

[Das et al., 2018a] Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018a). Embodied Question Answering. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[Das et al., 2018b] Das, A., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018b). Neural modular control for embodied question answering. In *Proceedings of the 2nd Annual Conference on Robot Learning*, pages 53–62.

[Das et al., 2017] Das, A., Kottur, S., Moura, J. M., Lee, S., and Batra, D. (2017). Learning cooperative visual dialog agents with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2951–2960.

[Dayan and Hinton, 1993a] Dayan, P. and Hinton, G. E. (1993a). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 271–278.

[Dayan and Hinton, 1993b] Dayan, P. and Hinton, G. E. (1993b). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 271–271. Morgan Kaufmann Publishers.

[DeepMind, 2016] DeepMind (2016). DeepMind AI reduces google data centre cooling bill by 40. https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/. Accessed: 2018-09-03.

[Deisenroth and Rasmussen, 2011] Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: a model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 465–472. Omnipress.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE.

[Devin et al., 2018] Devin, C., Abbeel, P., Darrell, T., and Levine, S. (2018). Deep object-centric representations for generalizable robot learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

[Dhingra et al., 2017] Dhingra, B., Liu, H., Yang, Z., Cohen, W., and Salakhutdinov, R. (2017). Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1832–1846.

[dos Santos et al., 2015] dos Santos, C., Xiang, B., and Zhou, B. (2015). Classifying relations by ranking with convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 626–634.

[Dosovitskiy et al., 2017] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16.

[Doya et al., 2002] Doya, K., Samejima, K., Katagiri, K.-i., and Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Computation*, 14(6):1347–1369.

[Duan et al., 2016a] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016a). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338.

[Duan et al., 2016b] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016b). RL^2: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110.

[Elfes, 1987] Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265.

[Fang et al., 2019] Fang, K., Toshev, A., Fei-Fei, L., and Savarese, S. (2019). Scene memory transformer for embodied agents in long-horizon tasks. *IEEE Conference on Computer Vision and Pattern Recognition*.

[Farabet et al., 2013] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929.

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

[Finn et al., 2017a] Finn, C., Abbeel, P., and Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*.

[Finn et al., 2016] Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72.

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

[Finn et al., 2017b] Finn, C., Yu, T., Fu, J., Abbeel, P., and Levine, S. (2017b). Generalizing skills with semi-supervised reinforcement learning. *International Conference on Learning Representations*.

[Foerster et al., 2016] Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145.

[Foerster et al., 2018a] Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. (2018a). Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, pages 122–130. International Foundation for Autonomous Agents and Multiagent Systems.

[Foerster et al., 2017] Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., and Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1146–1155. JMLR. org.

[Foerster et al., 2018b] Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018b). Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Foo et al., 2005] Foo, P., Warren, W. H., Duchon, A., and Tarr, M. J. (2005). Do humans integrate routes into a cognitive map? map-versus landmark-based navigation of novel shortcuts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 31(2):195.

[Fox et al., 2005] Fox, D., Thrun, S., and Burgard, W. (2005). *Probabilistic Robotics*. MIT press.

[Frank and Goodman, 2012] Frank, M. C. and Goodman, N. D. (2012). Predicting pragmatic reasoning in language games. *Science*, 336(6084):998–998.

[Fried et al., 2018] Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L.-P., Berg-Kirkpatrick, T., Saenko, K., Klein, D., and Darrell, T. (2018). Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, pages 3318–3329.

[Fukushima, 1979] Fukushima, K. (1979). Neural network model for a mechanism of pattern recognition unaffected by shift in position- neocognitron. *Transactions of the IECE*, J62-A(10):658–665.

[Gandhi et al., 2017] Gandhi, D., Pinto, L., and Gupta, A. (2017). Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955. IEEE.

[Gao et al., 2018] Gao, C., Mueller, M., and Hayward, R. (2018). Adversarial policy gradient for alternating markov games. *International Conference on Learning Representations, Workshop Track*.

[Giusti et al., 2016] Giusti, A., Guzzi, J., Cireşan, D. C., He, F., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G. D., Scaramuzza, D., and Gambardella, L. M. (2016). A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667.

[Gleave et al., 2019] Gleave, A., Dennis, M., Kant, N., Wild, C., Levine, S., and Russell, S. (2019). Adversarial policies: Attacking deep reinforcement learning. *CoRR*, abs/1905.10615.

[Goodfellow et al., 2014a] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

[Goodfellow et al., 2014b] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

[Grau-Moya et al., 2018] Grau-Moya, J., Leibfried, F., and Bou-Ammar, H. (2018). Balancing two-player stochastic games with soft q-learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 268–274. AAAI Press.

[Grosse et al., 2012] Grosse, R., Salakhutdinov, R. R., Freeman, W. T., and Tenenbaum, J. B. (2012). Exploiting compositionality to explore a large space of model structures. In *28th Conference on Uncertainly in Artificial Intelligence*, pages 15–17. AUAI Press.

[Gu et al., 2015] Gu, S., Ghahramani, Z., and Turner, R. E. (2015). Neural adaptive sequential Monte Carlo. In *Advances in Neural Information Processing Systems*, pages 2629–2637.

[Guo et al., 2014] Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems*, pages 3338–3346.

[Guo et al., 2016] Guo, X., Singh, S., Lewis, R., and Lee, H. (2016). Deep learning for reward design to improve monte carlo tree search in atari games. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1519–1525. AAAI Press.

[Gupta et al., 2017a] Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017a). Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer.

[Gupta et al., 2017b] Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017b). Cognitive mapping and planning for visual navigation. *IEEE Conference on Computer Vision and Pattern Recognition*.

[Gupta et al., 2017c] Gupta, S., Fouhey, D., Levine, S., and Malik, J. (2017c). Unifying map and landmark based representations for visual navigation. *CoRR*, abs/1712.08125.

[Haarnoja et al., 2017] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1352–1361. JMLR. org.

[He et al., 2016a] He, H., Boyd-Graber, J., Kwok, K., and Daumé III, H. (2016a). Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pages 1804–1813.

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of IEEE International Conference on Computer Vision*, pages 2980–2988. IEEE.

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[Heess et al., 2013] Heess, N., Tarlow, D., and Winn, J. (2013). Learning to pass expectation propagation messages. In *Advances in Neural Information Processing Systems*, pages 3219–3227.

[Heess et al., 2015] Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952.

[Higgins et al., 2017] Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1480–1490.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[Hoffmann et al., 2011] Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., and Weld, D. S. (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 541–550. Association for Computational Linguistics.

[Hu and Wellman, 1998a] Hu, J. and Wellman, M. P. (1998a). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250. Morgan Kaufmann Publishers Inc.

[Hu and Wellman, 1998b] Hu, J. and Wellman, M. P. (1998b). Online learning about other agents in a dynamic multiagent system. In *Proceedings of the Second International Conference on Autonomous Agents*, AGENTS '98, pages 239–246, New York, NY, USA. ACM.

[Hu et al., 2017] Hu, R., Rohrbach, M., Andreas, J., Darrell, T., and Saenko, K. (2017). Modeling relationships in referential expressions with compositional modular networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1115–1124.

[Ilin et al., 2007] Ilin, R., Kozma, R., and Werbos, P. J. (2007). Efficient learning in cellular simultaneous recurrent neural networks-the case of maze navigation problem. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 324–329. IEEE.

[Iyyer et al., 2015] Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 1681–1691.

[Jaderberg et al., 2017] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. *International Conference on Learning Representations*.

[Jain and Neal, 2004] Jain, S. and Neal, R. M. (2004). A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1):158–182.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

[Jang et al., 2017] Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with Gumbel-Softmax. *International Conference on Learning Representations*.

[Jiang et al., 2016] Jiang, X., Wang, Q., Li, P., and Wang, B. (2016). Relation extraction with multi-instance multi-label convolutional neural networks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1471–1480.

[Johnson et al., 2017] Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. (2017). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910.

[Johnson et al., 2015] Johnson, J., Krishna, R., Stark, M., Li, L.-J., Shamma, D., Bernstein, M., and Fei-Fei, L. (2015). Image retrieval using scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3668–3678.

[Johnson et al., 2016] Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 4246–4247.

[Joseph et al., 2013] Joseph, J., Geramifard, A., Roberts, J. W., How, J. P., and Roy, N. (2013). Reinforcement learning with misspecified model classes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 939–946. IEEE.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

[Kansky et al., 2017] Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., and George, D. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1809–1818. JMLR. org.

[Karl Moritz Hermann and PhilBlunsom, 2017] Karl Moritz Hermann, Felix Hill, S. G. F. W. R. F. H. S. D. S. W. M. C. M. J. D. T. M. W. C. A. D. H. and PhilBlunsom (2017). Grounded language learning in a simulated 3D world. *CoRR*, abs/1706.06551.

[Kemp and Tenenbaum, 2008] Kemp, C. and Tenenbaum, J. B. (2008). The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692.

[Kempka et al., 2016] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *CoRR*, abs/1312.6114.

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

[Kolve et al., 2017] Kolve, E., Mottaghi, R., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. (2017). AI2-THOR: An interactive 3D environment for visual AI. *CoRR*, abs/1712.05474.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

[Kulkarni et al., 2015] Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., and Mansinghka, V. (2015). Picture: A probabilistic programming language for scene perception. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399.

[Kulkarni et al., 2016] Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683.

[Kurutach et al., 2018] Kurutach, T., Tamar, A., Yang, G., Russell, S. J., and Abbeel, P. (2018). Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pages 8747–8758.

[Lake et al., 2015] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

[Lanctot et al., 2017] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Perolat, J., Silver, D., and Graepel, T. (2017). A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203.

[Larochelle et al., 2008] Larochelle, H., Erhan, D., and Bengio, Y. (2008). Zero-data learning of new tasks. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 2, pages 646–651. AAAI Press.

[Lauer and Riedmiller, 2000] Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann.

[Lazaridou et al., 2017] Lazaridou, A., Peysakhovich, A., and Baroni, M. (2017). Multi-agent cooperation and the emergence of (natural) language. *International Conference on Learning Representations*.

[Le et al., 2017] Le, T. A., Baydin, A. G., and Wood, F. (2017). Inference compilation and universal probabilistic programming. In *Artificial Intelligence and Statistics (AISTATS)*, pages 1338–1348.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Leibo et al., 2017] Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J., and Graepel, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems.

[Leonard and Durrant-Whyte, 1992] Leonard, J. J. and Durrant-Whyte, H. F. (1992). *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publishers, Norwell, MA, USA.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[Li and Malik, 2017] Li, K. and Malik, J. (2017). Learning to optimize neural nets. *CoRR*, abs/1703.00441.

[Li et al., 2017a] Li, R., Tapaswi, M., Liao, R., Jia, J., Urtasun, R., and Fidler, S. (2017a). Situation recognition with graph neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4173–4182.

[Li et al., 2019] Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., and Russell, S. (2019). Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[Li et al., 2017b] Li, Y., Cohn, T., and Baldwin, T. (2017b). Robust training under linguistic adversity. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

[Liang et al., 2008] Liang, P., Daumé III, H., and Klein, D. (2008). Structure compilation: trading structure for features. In *Proceedings of the 25th International Conference on Machine Learning*, pages 592–599. ACM.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

[Lin et al., 2016] Lin, Y., Shen, S., Liu, Z., Luan, H., and Sun, M. (2016). Neural relation extraction with selective attention over instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2124–2133.

[Littman, 1994] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, volume 157, pages 157–163.

[Liu et al., 2016] Liu, A., Soderland, S., Bragg, J., Lin, C. H., Ling, X., and Weld, D. S. (2016). Effective crowd annotation for relation extraction. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 897–906.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

[Lowe et al., 2017] Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*.

[Mahajan et al., 2018] Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision*, pages 181–196.

[Mandlekar et al., 2017] Mandlekar, A., Zhu, Y., Garg, A., Fei-Fei, L., and Savarese, S. (2017). Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE.

[Marino et al., 2017] Marino, K., Salakhutdinov, R., and Gupta, A. (2017). The more you know: Using knowledge graphs for image classification. *IEEE Conference on Computer Vision and Pattern Recognition*.

[Mateescu et al., 2010] Mateescu, R., Kask, K., Gogate, V., and Dechter, R. (2010). Join-graph propagation algorithms. *Journal of Artificial Intelligence Research*, 37(1):279–328.

[Matignon et al., 2012a] Matignon, L., Jeanpierre, L., and Mouaddib, A.-I. (2012a). Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI Conference on Artificial Intelligence*.

[Matignon et al., 2007] Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2007). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 64–69. IEEE.

[Matignon et al., 2012b] Matignon, L., Laurent, G. J., and Le Fort-Piat, N. (2012b). Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(01):1–31.

[McCarthy et al., 2006] McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (2006). A proposal for the Dartmouth summer research project on Artificial Intelligence, August 31, 1955. *AI Magazine*, 27(4):12–12.

[McCormac et al., 2017] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2017). SceneNet RGB-D: Can 5m synthetic images beat generic ImageNet pre-training on indoor segmentation? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2678–2687.

[Miller et al., 2017] Miller, A., Feng, W., Batra, D., Bordes, A., Fisch, A., Lu, J., Parikh, D., and Weston, J. (2017). ParlAI: A dialog research software platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 79–84.

[Mintz et al., 2009] Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011. Association for Computational Linguistics.

[Mirowski et al., 2018] Mirowski, P., Grimes, M., Malinowski, M., Hermann, K. M., Anderson, K., Teplyashin, D., Simonyan, K., kavukcuoglu, k., Zisserman, A., and Hadsell, R. (2018). Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, pages 2419–2430.

[Mirowski et al., 2017] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. (2017). Learning to navigate in complex environments. *International Conference on Learning Representations*.

[Mishkin et al., 2019] Mishkin, D., Dosovitskiy, A., and Koltun, V. (2019). Benchmarking classic and learned navigation in complex 3D environments. *CoRR*, abs/1901.10915.

[Mishra et al., 2018] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. *International Conference on Learning Representations*.

[Misra et al., 2017] Misra, D., Langford, J., and Artzi, Y. (2017). Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1004–1015.

[Miyato et al., 2017] Miyato, T., Dai, A. M., and Goodfellow, I. (2017). Adversarial training methods for semi-supervised text classification. *International Conference on Learning Representations*.

[Miyato et al., 2015] Miyato, T., Maeda, S.-i., Koyama, M., Nakae, K., and Ishii, S. (2015). Distributional smoothing with virtual adversarial training. *CoRR*, abs/1507.00677.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Moore and Russell, 2017] Moore, D. A. and Russell, S. J. (2017). Signal-based Bayesian seismic monitoring. *Artificial Intelligence and Statistics (AISTATS)*.

[Mordatch and Abbeel, 2018] Mordatch, I. and Abbeel, P. (2018). Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Morimoto and Doya, 2005] Morimoto, J. and Doya, K. (2005). Robust reinforcement learning. *Neural Computation*, 17(2):335–359.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814.

[Narasimhan et al., 2018] Narasimhan, K., Barzilay, R., and Jaakkola, T. (2018). Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874.

[Neal, 2011] Neal, R. M. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11).

[Neu and Szepesvári, 2007] Neu, G. and Szepesvári, C. (2007). Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 295–302. AUAI Press.

[Nguyen and Grishman, 2015] Nguyen, T. H. and Grishman, R. (2015). Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 39–48.

[Nisan et al., 2007] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic game theory*. Cambridge University Press.

[Nishihara et al., 2013] Nishihara, R., Minka, T., and Tarlow, D. (2013). Detecting parameter symmetries in probabilistic models. *CoRR*, abs/1312.5386.

[Nogueira and Cho, 2016] Nogueira, R. and Cho, K. (2016). End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pages 1903–1911.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871.

[Oh et al., 2017] Oh, J., Singh, S., Lee, H., and Kohli, P. (2017). Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2661–2670.

[Omidshafiei et al., 2017] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., and Vian, J. (2017). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2681–2690. JMLR. org.

[OpenAI et al., 2019a] OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. (2019a). Solving Rubik's cube with a robot hand. *CoRR*, abs/1910.07113.

[OpenAI et al., 2019b] OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019b). Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680.

[Osborne, 2004] Osborne, M. J. (2004). *An introduction to game theory*. Oxford university press New York.

[Paige and Wood, 2016] Paige, B. and Wood, F. (2016). Inference networks for sequential Monte Carlo in graphical models. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *JMLR*.

[Palatucci et al., 2009] Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. In *Advances in Neural Information Processing Systems*, pages 1410–1418.

[Pan and Yang, 2009] Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Panait and Luke, 2005] Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

[Parisotto and Salakhutdinov, 2018] Parisotto, E. and Salakhutdinov, R. (2018). Neural map: Structured memory for deep reinforcement learning. *International Conference on Learning Representations*.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

[Patel et al., 2015] Patel, V. M., Gopalan, R., Li, R., and Chellappa, R. (2015). Visual domain adaptation: A survey of recent advances. *IEEE Signal Processing Magazine*, 32(3):53–69.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787.

[Pathak et al., 2018] Pathak, D., Mahmoudieh, P., Luo, G., Agrawal, P., Chen, D., Shentu, Y., Shelhamer, E., Malik, J., Efros, A. A., and Darrell, T. (2018). Zero-shot visual imitation. In *International Conference on Learning Representations*.

[Peng et al., 2017] Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., and Wang, J. (2017). Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games. *CoRR*, abs/1703.10069.

[Peng et al., 2018a] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018a). Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE.

[Peng et al., 2018b] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018b). Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

[Perolat et al., 2017] Perolat, J., Strub, F., Piot, B., and Pietquin, O. (2017). Learning nash equilibrium for general-sum markov games from batch data. In *Artificial Intelligence and Statistics (AISTATS)*, pages 232–241.

[Pinto et al., 2017] Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2817–2826. JMLR. org.

[Pratt, 1993] Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. In *Advances in Neural Information Processing Systems*, pages 204–211.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

[Riedel et al., 2010] Riedel, S., Yao, L., and McCallum, A. (2010). Modeling relations and their mentions without labeled text. *Machine Learning and Knowledge Discovery in Databases*, pages 148–163.

[Riedmiller et al., 2018] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing-solving sparse reward tasks from scratch. *International Conference on Machine Learning*.

[Ritchie et al., 2015] Ritchie, D., Lin, S., Goodman, N. D., and Hanrahan, P. (2015). Generating design suggestions under tight constraints with gradient-based probabilistic programming. In *Computer Graphics Forum*, volume 34. Wiley Online Library.

[Ritchie et al., 2016] Ritchie, D., Thomas, A., Hanrahan, P., and Goodman, N. (2016). Neurally-guided procedural models: Amortized inference for procedural graphics programs using neural networks. In *Advances in Neural Information Processing Systems*, pages 622–630.

[Ross et al., 2011a] Ross, S., Gordon, G., and Bagnell, A. (2011a). A reduction of imitation learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*.

[Ross et al., 2011b] Ross, S., Munoz, D., Hebert, M., and Bagnell, J. A. (2011b). Learning message-passing inference machines for structured prediction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2737–2744. IEEE.

[Ross and Pineau, 2008] Ross, S. and Pineau, J. (2008). Model-based bayesian reinforcement learning in large structured domains. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 476–483. AUAI Press.

[Russell and Norvig, 2016] Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: a modern approach*. Pearson Education Limited.

[Rusu et al., 2017] Rusu, A. A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2017). Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 262–270.

[Sadeghi and Levine, 2017] Sadeghi, F. and Levine, S. (2017). CAD^{2}RL: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Savinov et al., 2018] Savinov, N., Dosovitskiy, A., and Koltun, V. (2018). Semi-parametric topological memory for navigation. *International Conference on Learning Representations*.

[Savinov et al., 2019] Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. (2019). Episodic curiosity through reachability. In *International Conference on Learning Representations*.

[Savva et al., 2017] Savva, M., Chang, A. X., Dosovitskiy, A., Funkhouser, T., and Koltun, V. (2017). MINOS: Multimodal indoor simulator for navigation in complex environments. *CoRR*, abs/1712.03931.

[Schaul and Schmidhuber, 2010] Schaul, T. and Schmidhuber, J. (2010). Metalearning. *Scholarpedia*, 5(6):4650.

[Schmidhuber, 1990] Schmidhuber, J. (1990). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *International Joint Conference on Neural Networks*. IEEE.

[Schmidhuber, 1995] Schmidhuber, J. (1995). On learning how to learn learning strategies.

[Schrittwieser et al., 2019] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by planning with a learned model. *CoRR*, abs/1911.08265.

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

[Shi et al., 2017] Shi, T., Karpathy, A., Fan, L., Hernandez, J., and Liang, P. (2017). World of Bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395.

[Søgaard, 2013] Søgaard, A. (2013). Part-of-speech tagging with antagonistic adversaries. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 640–644.

[Song et al., 2017a] Song, J., Zhao, S., and Ermon, S. (2017a). A-NICE-MC: Adversarial training for MCMC. In *Advances in Neural Information Processing Systems*, pages 5140–5150.

[Song et al., 2017b] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017b). Semantic scene completion from a single depth image. *IEEE Conference on Computer Vision and Pattern Recognition*.

[Spiegelhalter et al., 1996] Spiegelhalter, D. J., Thomas, A., Best, N. G., Gilks, W., and Lunn, D. (1996). BUGS: Bayesian inference using Gibbs sampling. *Version 0.5,(version ii) http://www. mrc-bsu. cam. ac. uk/bugs*, 19.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Stern et al., 2009] Stern, D. H., Herbrich, R., and Graepel, T. (2009). Matchbox: large scale online Bayesian recommendations. In *Proceedings of the 18th International Conference on World Wide Web*, pages 111–120. ACM.

[Stuhlmüller et al., 2013] Stuhlmüller, A., Taylor, J., and Goodman, N. (2013). Learning stochastic inverses. In *Neural Information Processing Systems*.

[Sukhbaatar et al., 2018] Sukhbaatar, S., Kostrikov, I., Szlam, A., and Fergus, R. (2018). Intrinsic motivation and automatic curricula via asymmetric self-play. *International Conference on Learning Representations*.

[Sukhbaatar et al., 2016] Sukhbaatar, S., szlam, a., and Fergus, R. (2016). Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252.

[Surdeanu et al., 2012] Surdeanu, M., Tibshirani, J., Nallapati, R., and Manning, C. D. (2012). Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465. Association for Computational Linguistics.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

[Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

[Synnaeve et al., 2016] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N. (2016). TorchCraft: a library for machine learning research on real-time strategy games. *CoRR*, abs/1611.00625.

[Szegedy et al., 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, abs/1312.6199.

[Tai and Liu, 2016] Tai, L. and Liu, M. (2016). Towards cognitive exploration through deep reinforcement learning for mobile robots. *CoRR*, abs/1610.01733.

[Tamar et al., 2016] Tamar, A., Wu, Y., Thomas, G., Levine, S., and Abbeel, P. (2016). Value iteration networks. In *Advances in Neural Information Processing Systems*.

[Tampuu et al., 2017] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395.

[Tan, 1993] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337.

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

[Tesauro, 2004] Tesauro, G. (2004). Extending q-learning to general adaptive multi-agent systems. In *Advances in Neural Information Processing Systems*, pages 871–878.

[Thomas and Barto, 2011] Thomas, P. S. and Barto, A. G. (2011). Conjugate markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning*, pages 137–144.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT press.

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

[Tian et al., 2017] Tian, Y., Gong, Q., Shang, W., Wu, Y., and Zitnick, C. L. (2017). ELF: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*, pages 2659–2669.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5. COURSERA: Neural Networks for Machine Learning.

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

[Torralba et al., 2003] Torralba, A., Murphy, K. P., Freeman, W. T., and Rubin, M. A. (2003). Context-based vision system for place and object recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, page 273. IEEE Computer Society.

[Turek et al., 2016] Turek, D., de Valpine, P., Paciorek, C. J., and Anderson-Bergman, C. (2016). Automated parameter blocking for efficient Markov chain Monte Carlo sampling. *Bayesian Analysis*.

[Uther and Veloso, 1997] Uther, W. T. and Veloso, M. M. (1997). Generalizing adversarial reinforcement learning. In *Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems*, page 206. Citeseer.

[Vedantam et al., 2019] Vedantam, R., Desai, K., Lee, S., Rohrbach, M., Batra, D., and Parikh, D. (2019). Probabilistic neural-symbolic models for interpretable visual question answering. In *International Conference on Machine Learning*.

[Venugopal and Gogate, 2013] Venugopal, D. and Gogate, V. (2013). Dynamic blocking and collapsing for Gibbs sampling. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 664–673. AUAI Press.

[Vilalta and Drissi, 2002] Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95.

[Vinyals et al., 2019] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[Vinyals et al., 2017] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. (2017). Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782.

[Wang et al., 2016] Wang, L., Cao, Z., De Melo, G., and Liu, Z. (2016). Relation classification via multi-level attention CNNs. In *Proceedings of the 54th annual meeting of the Association for Computational Linguistics*, pages 1298–1307.

[Wang and Spelke, 2002] Wang, R. F. and Spelke, E. S. (2002). Human spatial representation: Insights from animals. *Trends in Cognitive Sciences*, 6(9):376–382.

[Wang et al., 2018a] Wang, T., Wu, Y., Moore, D., and Russell, S. J. (2018a). Meta-learning MCMC proposals. In *Advances in Neural Information Processing Systems*.

[Wang and Russell, 2015] Wang, W. and Russell, S. (2015). A smart-dumb/dumb-smart algorithm for efficient split-merge MCMC. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 902–911. AUAI Press.

[Wang et al., 2018b] Wang, X., Xiong, W., Wang, H., and Yang Wang, W. (2018b). Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision*, pages 37–53.

[Wang et al., 2018c] Wang, X., Ye, Y., and Gupta, A. (2018c). Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6857–6866.

[Watter et al., 2015] Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754.

[Wei et al., 2018] Wei, E., Wicke, D., Freelan, D., and Luke, S. (2018). Multiagent soft Q-learning. In *2018 AAAI Spring Symposium Series*.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

[Wu et al., 2017a] Wu, C., Kreidieh, A., Parvate, K., Vinitsky, E., and Bayen, A. M. (2017a). Flow: Architecture and benchmarking for reinforcement learning in traffic control. *CoRR*, abs/1710.05465.

[Wu et al., 2016a] Wu, Q., Wang, P., Shen, C., Dick, A., and van den Hengel, A. (2016a). Ask me anything: Free-form visual question answering based on knowledge from external sources.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4622–4630.

[Wu et al., 2017b] Wu, Y., Bamman, D., and Russell, S. (2017b). Adversarial training for relation extraction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

[Wu et al., 2016b] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016b). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

[Wu et al., 2018] Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building generalizable agents with a realistic and rich 3D environment. *International Conference on Learning Representations, Workshop Track*.

[Wu et al., 2019] Wu, Y., Wu, Y., Tamar, A., Russell, S., Gkioxari, G., and Tian, Y. (2019). Bayesian relational memory for semantic visual navigation. In *Proceedings of the IEEE International Conference on Computer Vision*.

[Xie et al., 2017] Xie, Z., Wang, S. I., Li, J., Lévy, D., Nie, A., Jurafsky, D., and Ng, A. Y. (2017). Data noising as smoothing in neural network language models. *International Conference on Learning Representations*.

[Xu et al., 2017] Xu, H., Gao, Y., Yu, F., and Darrell, T. (2017). End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2174–2182.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

[Yang et al., 2019] Yang, W., Wang, X., Farhadi, A., Gupta, A., and Mottaghi, R. (2019). Visual semantic navigation using scene priors. In *International Conference on Learning Representations*.

[Yang et al., 2018] Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. (2018). Mean field multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5567–5576.

[Zambaldi et al., 2019] Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., and Battaglia, P. (2019). Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations*.

[Zeng et al., 2015] Zeng, D., Liu, K., Chen, Y., and Zhao, J. (2015). Distant supervision for relation extraction via piecewise convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1753–1762.

[Zeng et al., 2014] Zeng, D., Liu, K., Lai, S., Zhou, G., and Zhao, J. (2014). Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

[Zhang et al., 2018] Zhang, A., Sukhbaatar, S., Lerer, A., Szlam, A., and Fergus, R. (2018). Composable planning with attributes. In *International Conference on Machine Learning*, pages 5837–5846.

[Zhang et al., 2017] Zhang, H., Kyaw, Z., Chang, S.-F., and Chua, T.-S. (2017). Visual translation embedding network for visual relation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5532–5540.

[Zhang et al., 2019] Zhang, H., Zhou, H., Miao, N., and Li, L. (2019). Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569.

[Zhou et al., 2016] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. (2016). Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of The 54th Annual Meeting of the Association for Computational Linguistics*, page 207.

[Zhu et al., 2017a] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017a). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232.

[Zhu et al., 2017b] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017b). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE.

[Zhu et al., 2015] Zhu, Y., Urtasun, R., Salakhutdinov, R., and Fidler, S. (2015). segdeepm: Exploiting segmentation and context in deep neural networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4703–4711.