Transfer in Sequential Decision Making

Jelena Luketina

Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2023

Abstract

Transfer learning is critical for improving the data efficiency and applicability of deep learning models in sequential decision-making. However, determining what knowledge transfers and how to effectively leverage it remains an open challenge. Recent breakthroughs in representation learning, especially in language and vision domains, demonstrate the power of transfer from large-scale datasets. Meanwhile, progress in simulation platforms and environment designs has opened up new possibilities for collecting diverse, realistic training data. Against this backdrop, the four works contained in this thesis explore transfer techniques in various aspects of sequential decision-making.

First, we provide a comprehensive survey of prior work on integrating natural language data and representations in sequential decision-making. Our survey reveals open challenges and charts promising research directions, advocating for the greater utilization of large language models and development of more semantically complex environments. Second, we propose and study a modular architecture design for compositional generalization in multi-modal multi-task settings. Controlled experiments demonstrate zero-shot transfer on held-out compositions of observation, action and instruction spaces, as well as efficient integration of new observation modalities. Third, we propose a method for directing unsupervised skill discovery toward more useful behaviors by transferring knowledge about value-relevant state features from the source tasks. Experiments in continuous control domains show our method yields superior coverage of the relevant dimensions of the state space and improved performance on the downstream tasks. Finally, our analysis of meta-gradients in non-stationary environments demonstrates that learning optimizers as functions of contextual features enables faster adaptation and increased lifetime performance.

Overall, the thesis offers novel insights and strategies for effective knowledge transfer in sequential decision-making. The works illustrate the benefits of incorporating language, targeted inductive biases, modest supervision, and meta-learned adaptation.

Transfer in Sequential Decision Making

Jelena Luketina
Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2023

Acknowledgements

I would like to express my deep gratitude to my supervisor, Shimon Whiteson, whose guidance, support, and feedback have been invaluable throughout my PhD journey. Under his mentorship, I have grown as a scientist, learned to refine my ideas and sharpen my writing.

Throughout my PhD, I have had the privilege of being part of two vibrant academic communities: WhiRL as a member, and UCL DARK as a frequent visitor and collaborator. Both research labs have been home to immensely talented individuals who have shaped my development. I am particularly indebted to my collaborators, without whom this thesis would not have been possible: Matt Smith, Tim Rocktäschel, Jakob Foerster, Robert Kirk, Minqi Jiang, Nantas Nardelli, Maximilian Igl, Kristian Hartikainen, Greg Farquhar, Wendelin Boehmer, Pasquale Minervini, Roberta Raileanu, and Ed Grefenstette. Thank you for the countless insightful discussions, productive coding sessions, and memorable late-night deadline sprints. To my other colleagues at WhiRL and UCL DARK – Luisa Zintgraf, Christian Schroeder de Witt, Akbir Khan, Laura Ruis, Vitaly Kurin, Matthew Fellows, Risto Vuorio, Anuj Mahajan, Kamil Ciosek, Supratik Paul, Tabish Rashid, Mingfei Sun, Bei Peng, Shangtong Zhang, and Patrick Lewis – thank you for creating an enriching intellectual environment and making this journey more enjoyable!

My research experience was further enriched through internships at Google DeepMind and Meta. I am especially grateful to my exceptional managers and mentors, Tom Zahavy and Arthur Szlam, both for taking a chance on me and for teaching me how to do great research. My sincere thanks extend to my industry colleagues and collaborators: Sebastian Flennerhag, Jack Lanchantin, Sainbayar Sukhbaatar, Yannick Schroecker, David Abel, Ishita Mediratta, Christoforos Nalmpantis, Satinder Singh, Eric Hambro, and Roberta Raileanu. Thank you all for making me feel welcomed and supported, even when we did not get to work in-person.

Finally, I would like to express my appreciation to Tom Rainforth and Karthik Narasimhan for generously offering their time and expertise in examining this thesis.

Abstract

Transfer learning is critical for improving the data efficiency and applicability of deep learning models in sequential decision-making. However, determining what knowledge transfers and how to effectively leverage it remains an open challenge. Recent breakthroughs in representation learning, especially in language and vision domains, demonstrate the power of transfer from large-scale datasets. Meanwhile, progress in simulation platforms and environment designs has opened up new possibilities for collecting diverse, realistic training data. Against this backdrop, the four works contained in this thesis explore transfer techniques in various aspects of sequential decision-making.

First, we provide a comprehensive survey of prior work on integrating natural language data and representations in sequential decision-making. Our survey reveals open challenges and charts promising research directions, advocating for the greater utilization of large language models and development of more semantically complex environments. Second, we propose and study a modular architecture design for compositional generalization in multi-modal multi-task settings. Controlled experiments demonstrate zero-shot transfer on held-out compositions of observation, action and instruction spaces, as well as efficient integration of new observation modalities. Third, we propose a method for directing unsupervised skill discovery toward more useful behaviors by transferring knowledge about value-relevant state features from the source tasks. Experiments in continuous control domains show our method yields superior coverage of the relevant dimensions of the state space and improved performance on the downstream tasks. Finally, our analysis of meta-gradients in non-stationary environments demonstrates that learning optimizers as functions of contextual features enables faster adaptation and increased lifetime performance.

Overall, the thesis offers novel insights and strategies for effective knowledge transfer in sequential decision-making. The works illustrate the benefits of incorporating language, targeted inductive biases, modest supervision, and meta-learned adaptation.

Contents

List of Figures

List of Abbreviations

AC . . . . . Actor-Critic.

BC . . . . . Behavioral Cloning.

BERT . . . . . Bidirectional Encoder Representations from Transformers.

BMG . . . . . Bootstrapped Meta Gradients

COIN . . . . . Compositional Interfaces.

DIAYN . . . . Diversity is All You Need.

DIVRS . . . . Diverse in Value-Relevant Space.

DRL . . . . . Deep Reinforcement Learning.

GPT . . . . . Generative Pre-trained Transformer.

HRL . . . . . Hierarchical Reinforcement Learning.

IL . . . . . Imitation Learning.

IRL . . . . . Inverse Reinforcement Learning.

LLM . . . . . Large Language Model.

MDP . . . . . Markov Decision Process.

MG . . . . . Meta Gradients.

MI . . . . . Mutual Information.

NLP . . . . . Natural Language Processing.

PG . . . . . Policy Gradient.

PPO . . . . . Proximal Policy Optimization.

RL . . . . . Reinforcement Learning.

SAC . . . . . Soft Actor-Critic.

SDM . . . . . Sequential Decision Making.

SGD . . . . . Stochastic Gradient Descent.

TD . . . . . Temporal Difference.

VLM . . . . . Vision-Language Model.

VOD . . . . . Variational Option Discovery.

1 Introduction

Contents

1.1 Motivation

Transfer learning, the ability to apply knowledge gained in one setting to a novel setting, is one of the hallmarks of human intelligence. We are able to detect deep structural similarities between diverse tasks and generalize beyond superficial differences, enabling us to rapidly learn and adapt to new and often disparate scenarios (Singley & Anderson, 1989; Lobato, 2006). Consider as an example Surya Bonaly, a professional athlete whose initial experiences in the field of gymnastics were followed by quick mastery and very successful career in figure skating (Murphy & Bonaly, 2022). Her swift career pivot illustrates the targeted application of overlapping skill sets – an acute balance and fine-tuned motor control – skills both critical in gymnastics and figure skating. In contrast, machine learning systems tend to be narrow in their abilities, requiring extensive training to master each domain, while their performance can deteriorate with even slight changes from the

training distribution. Similar transfer abilities in artificially intelligent systems would open them up to much wider application areas.

Within machine learning, transfer is conceptualized as an improvement in the performance of learners in *target* domains by utilizing knowledge obtained by training on *source* domains (Caruana, 1997; Pan & Yang, 2009). This notion is especially important in deep learning, as such models are extremely data-demanding, often requiring millions of data samples to solve the task. In fact, some of the biggest success stories of deep learning rely on transfer from domains where the data is plentiful, to related domains in which the data is scarce. For example, models pre-trained on ImageNet dataset (Deng et al., 2009) require significantly less data to adapt to new visual tasks, such as image segmentation (Oquab et al., 2014; He et al., 2017) or classification in narrower domains (Yosinski et al., 2014; Xie & Richmond, 2018). Similarly, transfer learning underpins recent breakthroughs in natural language processing, with models first pre-trained on vast amounts of text through self-supervision (Devlin et al., 2018; Brown et al., 2020). The knowledge encoded in the parameters of these models can then be efficiently fine-tuned for a wide range of language tasks, including summarization (Stiennon et al., 2020) or sentiment analysis (Sun et al., 2019).

An important class of problems benefiting from transfer are sequential decision-making (SDM), where a trained agent is tasked with making decisions based on successive observations in order to maximize rewards over time. These include important applications in robotics, finance, dialogue systems, and more (Arulkumaran et al., 2017). Sequential decision-making problems are typically formulated as Markov decision processes (MDPs). When expert data is available, such problems can be solved with imitation learning (IL), resulting in policies that mimic expert behavior (Argall et al., 2009). On the other hand, when the agent is able to interact with the environment during training, reinforcement learning (RL) becomes a viable approach (Sutton & Barto, 2018). In many highly challenging applications – including games like Go and Starcraft, as well as chatbots, the two approaches are

combined. An agent is first trained through imitation, followed by fine-tuning with reinforcement learning (Silver et al., 2016; Vinyals et al., 2019; Ouyang et al., 2022).

Transfer learning is particularly relevant for SDM: the data requirements tend to be particularly high due as the stochasticity and complexity of the decision space, and the data collection is challenging. As an example, let us consider robotics. The real-world interaction is slow, involves costly maintenance of equipment, and poses risks of damage to the robot and its environment (Ibarz et al., 2021). When the real-world environment can be simulated with sufficiently high fidelity, a common transfer learning approach is sim-to-real (Peng et al., 2018): the policies are first trained in a simulated environment where the interaction is fast and safe, and then fine-tuned on the real robot. Speaking more generally, by utilizing transfer between related tasks, we can obtain agents that need not be trained from scratch for each individual task, but that can continually learn and accumulate skills over a lifetime (Schwarz et al., 2018).

While transfer learning holds great promise, what enables effective transfer in any particular domain is typically an empirical question. Naively applying transfer can result in negative transfer – a phenomenon where the performance on the target task is made worse by training on the source task (Rosenstein et al., 2005). The cause of negative transfer could be a lack of sufficient similarity between the source and target task (for example, transferring the weights of an agent trained on a racing game to a puzzle game), or the specific transferred knowledge is misleading or irrelevant for the target task (for example, transferring features responsible for object manipulation to a task involving only simple navigation) (Weiss et al., 2016). Hence it is essential for any novel transfer learning method to identify relevant source tasks and determine what aspect of agent or training on the source tasks could be utilized on the target tasks.

The contributions included in this thesis address a variety of challenges in SDM by relying on transfer learning: from leveraging natural language data, to enabling compositional generalization in multi-modal multi-task settings, learning meaningfully diverse sets of skills and adapting learning algorithms in non-stationary

environments. By exploring various transfer settings and methods, this thesis aims to provide insights into the conditions and techniques that enable effective transfer in sequential decision-making.

1.2 Contributions

In this section, we provide a high-level overview of four different works, each of which will form a Chapter of the thesis. The first contribution (Chapter 3) is a survey and position paper, whereas the remaining three contributions (Chapters 4, 5 and 6) develop new algorithms or provide an analysis of existing algorithms. The background for these works is provided in Chapter 2.

Reinforcement Learning Informed by Natural Language

Language representations pre-trained on large unlabeled corpora have become ubiquitous in natural language processing (NLP). By learning to predict words and sentences in a self-supervised manner, models like BERT and GPT acquire syntactic and semantic knowledge that transfers well to downstream NLP tasks (Devlin et al., 2018; Radford et al., 2019). The success of this transfer learning approach raises an intriguing question: can similar techniques help reinforcement learning agents acquire useful real-world knowledge and priors?

To facilitate further research in this direction, in Chapter 3, we survey the landscape of existing work on the integration of natural language into sequential decision-making. We categorize these efforts into two settings: language-conditional RL, where interaction with language is necessitated by the problem formulation itself; and language-assisted RL, where language provides additional guidance but is not strictly required for solving the task.

Language-conditional RL includes problems like instruction following, where agents must interpret natural language goal specifications (Section 3.2.1); problems in which rewards are derived from language annotations (Section 3.2.1); and environments like text games where language is built into the action and observation spaces (Section 3.2.1). The paper reviews work on grounding language in these

settings, often using techniques like parsing instructions into formal representations or embedding instructions and observations into a shared vector space. We observe that most of this work uses simplified synthetic languages rather than real human language and tasks are limited to narrow domains like navigation.

For language-assisted RL, textual resources provide supplementary knowledge to guide learning, without being essential to the core SDM problem. This includes transfer from task-specific manuals and corpora describing environmental dynamics and strategies, a largely unexplored area. Other works use language to shape internal representations and structure policies. Here again, the use of real-world natural language data is limited.

We argue that progress in representation learning for language warrants revisiting tighter integration of NLP and RL, such as using language models for planning or representation. Since the publication of our survey in 2019 (Luketina et al., 2019), the advances in large pre-trained language models (LLMs) have begun realizing some of this potential. For example, LLMs have been used as initializations of RL policies (Mu et al., 2022), for guiding exploration and shaping of reward functions (Li et al., 2022; Klissarov et al., 2023) and planning Huang et al. (2022c); Ahn et al. (2022). On the other hand, with projects such as AutoGPT (Gravitas, 2023) which provide integration with various APIs and maintenance of memory, LLMs are starting to be used as autonomous agents. These works reinforce our claims about the potential of language data in enabling more capable, generalizable and interpretable agents.

Our work also advocates for more research on language-assisted RL, the development of more complex and semantically rich environments, and standardized benchmarks to measure progress. Key open challenges include scaling grounding techniques and moving beyond closed worlds and simplified synthetic languages. As a promising approach to scalable grounding, recent works utilize pre-trained vision-language models (Zitkovich et al., 2023) in domains with visual observation spaces. Moreover, with the recent advances in simulators and testbeds (Juliani et al., 2020; Li et al., 2021; Yao et al., 2022), the research community may now have the tools to make rapid progress in this direction. Our suggestions around

developing more diverse environments with real-world semantics and standardized evaluations are still relevant for advancing research in this area.

Compositional Interfaces for Compositional Generalization

Training a single generalist agent that can accomplish a wide variety of tasks and perceive multiple modalities has many potential benefits (Reed et al., 2022): it could enable the transfer of common knowledge between the domains, improving generalization and fostering faster adaptation to new domains, while at the same time reducing the need for hand-crafting models with domain-specific inductive biases. However, how to best parameterize such generalist agents to effectively handle multiple modalities and enable positive transfer is still an open question.

In Chapter 4, we study these questions with a modular architecture through a set of controlled experiments. To separate and study individual research questions in isolation, as well as design enough tasks to mimic the regime of large-scale multi-task training, we develop an environment that supports mixing and matching observation modalities, action spaces, and instructions. The modular neural architecture we propose for addressing this setting consists of separate encoder and decoder networks for each observation and action space, with an in-between shared controller model.

Our experiments demonstrate that the proposed modular architecture generalizes effectively to held-out compositions of observations, actions, and instructions during evaluation. The controller transfers knowledge across domains while perception and action modules specialize to their respective spaces. The proposed architecture also enables efficient integration of new observation modalities through training of just the newly added perception modules. Overall, the paper provides compelling evidence that compositional generalization in generalist agents can be achieved by careful modular design.

The paper's approach of breaking down problems into reusable compositional interfaces is a promising general strategy for transfer learning. Developing modular agents that quickly adapt their functional components to new environments while reusing others is also an important direction for lifelong learning.

Learning Skills Diverse in Value-Relevant Features

For many sequential decision-making problems, it is useful to obtain skills – sequences of actions or sub-policies that correspond to abstract behaviors – such as grasping or walking gaits in robotics. In addition to capturing more easily transferable components of policies, skills also benefit training by structuring exploration and speeding up credit assignment. Constructing skills, however, is a significant challenge, as most practical methods rely on the hand-crafting of policies or subtasks. Information-theoretic methods (Eysenbach et al., 2019; Achiam et al., 2018) offer a promising alternative: a set of diverse skills is learned in a purely unsupervised manner through the environment interaction, by maximizing the discriminability of the states visited by individual skills. However, since only some features of the state matter in complex environments, these methods often discover behaviors that are trivially diverse, learning skills that are not helpful in downstream tasks.

In Chapter 5, we propose addressing this problem through transfer learning by utilizing a small set of representative source tasks to obtain a diversity metric. Our approach consists of two phases. First, by training on the source tasks we learn which features are the most predictive of returns. Then, the discriminability objective for an unsupervised information-theoretic method is defined on this learned feature space. This results in the construction of sets of skills that are both diverse and more useful by controlling only the most important features.

The intuition is that while some state features may be easy for skills to manipulate, they may not correspond to functionally meaningful behaviors. For example, in a simulated robotic system, directives to explore joint angle space may produce trivial motion. Instead, skills that cover different ambulation patterns or object interaction behaviors are likely to prove more useful for downstream tasks. Identifying a low-dimensional embedding focused on predicting state values helps restrict diversity to more useful feature combinations.

Experiments in continuous control tasks demonstrate that steering unsupervised skill discovery using value-relevant features substantially improves performance on downstream tasks compared with naive state space diversity. The learned skills

demonstrate qualitatively better coverage of state dimensions that matter for solving new tasks. The approach illustrates how modest amounts of supervision can help shape representations to direct unsupervised learning in more useful directions.

Meta-Gradients in Non-stationary Environments

The performance of deep reinforcement learning algorithms is highly sensitive to the choice of hyper-parameters like learning rates, exploration rates and discount factors Eimer et al. (2023). Moreover, the optimal values of such hyper-parameters are expected to change over the course of training. Adaptation of hyper-parameters is especially important in non-stationary environments, however, since the hyper-parameter schedules in such settings can not be known in advance, there is a need for methods capable of *learning* such schedules from experience.

Among different approaches to adaptive hyper-parameter schedules in SDM, meta-gradient methods have a particularly good track record Zahavy et al. (2020); Flennerhag et al. (2022). In Chapter 6, we provide a study of the behavior of such methods in non-stationary environments, specifically focusing on contextual meta-gradients: unlike typical meta-gradients which adapt only the value of the hyper-parameter, contextual meta-gradients learn hyper-parameters as *functions of selected features* of the agent's experience.

Experiments across two non-stationary environments demonstrate that contextual meta-gradients consistently improve performance over non-contextual meta-gradients and fixed hyperparameter values. The benefits are especially pronounced in rapidly changing environments, where fast adaptation is critical. We also find that more information in the context features generally leads to improved performance. These results are also supported by qualitative analysis of resulting hyper-parameter schedules and learned functions of context features.

Through this work, we shed light on a strategy for meta-learned transfer of optimization knowledge during a learning lifetime: by conditioning hyper-parameter predictions on contextual features, we allow for the transfer of knowledge about suitable hyper-parameter values between different phases of learning.

1.2.1 Published Works

Chapters 3 to 6 are based on the following published papers and pre-prints, with the star "*" indicating first or shared first authors:

- **Title**: A Survey of Reinforcement Learning Informed by Natural Language
  Authors: Jelena Luketina*, Nantas Nardelli, and Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, Tim Rocktäschel.
  Venue: IJCAI, 2019.

- **Title**: Learning Skills Diverse in Value-Relevant Features
  Authors: Matthew J. A. Smith*, Jelena Luketina*, Kristian Hartikainen, Maximilian Igl, Shimon Whiteson
  Venue: CoLLAs, 2022.

- **Title**: Meta-Gradients in Non-Stationary Environments
  Authors: Jelena Luketina*, Sebastian Flennerhag, Yannick Schroecker, David Abel, Tom Zahavy, Satinder Singh
  Venue: CoLLAs, 2022. (Oral)

- **Title**: Compositional Interfaces for Compositional Generalization
  Authors: Jelena Luketina*, Jack Lanchantin, Sainbayar Sukhbaatar, Arthur Szlam
  Venue: CoLLAs, 2024.

While working on this thesis, I also contributed to the following projects which have not been included in the thesis:

- **Title**: WordCraft: An Environment for Benchmarking Commonsense Agents
  Authors: Minqi Jiang*, Jelena Luketina*, Nantas Nardelli*, Pasquale Minervini, Philip H. S. Torr, Shimon Whiteson, Tim Rocktäschel
  Venue: Workshop on Language in Reinforcement Learning, ICML, 2020.

- **Title**: Transient Non-stationarity and Generalisation in Deep Reinforcement Learning

  Authors: Maximilian Igl*, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, Shimon Whiteson

  Venue: ICLR, 2020.

- **Title**: Understanding the Effects of RLHF on LLM Generalisation and Diversity

  Authors: Robert Kirk*, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, Roberta Raileanu

  Venue: ICLR, 2024.

2
Background

Contents

In this Chapter, we provide the background and introduce the formalism used throughout the rest of the thesis. The formalism of Markov Decision Processes (MDPs) is introduced in Section 2.1. The background and description of selected methods in Reinforcement Learning (RL) and Imitation Learning (IL) as approaches to solving MDPs, are given in Sections 2.2 and 2.3. Section 2.2.3 introduces the Hierarchical Reinforcement Learning (HRL) methods, which is relevant for Chapters 3 and 5. Natural Language Processing (NLP) methods like language models play

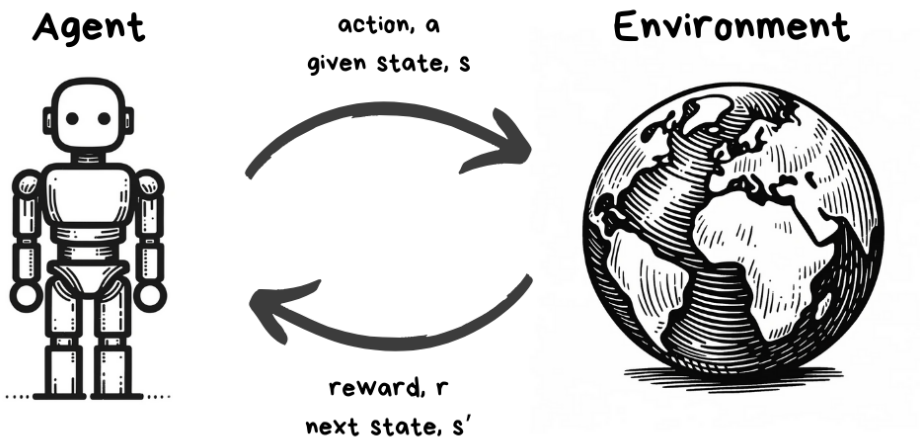


Figure 2.1: A simple illustration of agent and environment in a MDP: the agent selects an action a given state s, resulting in reward r and transition into a state s'.

a role in Chapters 3 and 4, and we provide their brief overview in Section 2.5. Lastly, Section 2.4 gives a background on meta-gradients in RL required for Chapter 6. Throughout this thesis, we presume a working knowledge of the fundamental concepts in deep learning. For detailed explanations and theoretical background on deep learning methodology, the reader is referred to Goodfellow et al. (2016).

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework commonly used in sequential decision-making to model an interaction of an agent with an environment (Sutton & Barto, 2018). As shown in Figure 2.1, the agent perceives the state of the environment s, upon which it selects an action a, resulting in a transition into a new state s' and a reward r. This entire process is defined by a tuple $\langle S, A, R, P, P_0, \gamma, H \rangle$ where:

- S is the set of states, representing all the possible configurations of the environment;
- A is the set of actions available to the agent;

- $R : S \times A \times S \to \mathbb{R}$ is the reward function, providing immediate feedback upon transitioning from state s to s' given action a;
- $P : S \times A \times S \to [0, 1]$ is the transition function, specifying the probability of transitioning from state s to s' given action a;
- $P_0 : S \to [0, 1]$ is the initial or starting state distribution;
- $\gamma \in [0, 1]$ is the discount factor, determining the present value of future rewards;
- H is the time horizon, which can be finite or infinite and defines the length of the interaction. When the horizon is infinite however, we require $\gamma \in [0, 1)$.

The Markov assumption is that distribution over the next state as well as the reward depends only on the current state and action, not on the sequence of events that preceded them. This memoryless property simplifies the decision-making model significantly. The agent starts in a state, and from there selects actions according to the policy $\pi(a|s) = p(A = a|S = s)$. The resulting sequence of states, actions, and rewards after T steps $\{(s_t, a_t, r_t, s_{t+1})\}_{t=t_0}^{t=T}$ is referred to as *rollout*. This sequence can be finite (ending when a terminal state or the horizon is reached) or infinite. A *terminal state* is a state that, once reached, results in no further transitions or rewards (e.g., the goal state in path finding or game-over screen in video games). An *episode* refers to a complete rollout from the initial to a terminal state. In the rest of this chapter, we assume an infinite horizon (i.e., $H \to \infty$), as such an assumption further simplifies the theoretical analysis of MDPs. Note that in the infinite horizon case, the discount factor has to be strictly less than one to ensure convergence. The simplification of analysis is also the reason why most RL algorithms assume an infinite horizon. Empirically this tends to be a good enough approximation even for problems with a finite horizon, with a discount factor managing the trade-off between immediate and future rewards.

The *value of a state* under policy is defined as an expected sum of discounted rewards r_t when starting from state s and following policy π :

$$V^{\pi}(s) = \mathbb{E}_{P,\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k | s_t = s\right], \tag{2.1}$$

where $V^{\pi}(s)$ is referred to as the state-value function. This discounted sum of rewards is also called a *return*.

Similarly, we can define an action-value function $Q^{\pi}(s, a)$ (also referred to as a Q-function) as an expected return starting from state s, taking action a, and thereafter following policy π:

$$Q^{\pi}(s, a) = \mathbb{E}_{P,\pi}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_k | s_t = s, a_t = a\right]. \tag{2.2}$$

The objective in a MDP is to find a policy π^* (referred to as *optimal policy*) that maximizes the expected discounted cumulative return:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{P,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{2.3}$$

2.1.1 The Bellman Equations

The Bellman equations and optimality conditions are essential for understanding and designing algorithms for sequential decision-making problems. They provide both a conceptual framework for thinking about the values of decisions and practical computational tools for finding optimal policies.

We start by deriving the Bellman equations from the recursive decomposition of value functions (2.1) and (2.2):

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s, a)\Big[R(s, a, s') + \gamma V^{\pi}(s')\Big] \tag{2.4}$$

$$Q^{\pi}(s, a) = \sum_{s' \in S} P(s'|s, a)\Big[R(s, a, s') + \gamma \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s', a')\Big]. \tag{2.5}$$

The values of optimal policy $V^*(s)$ at each state s satisfy $V^*(s) = \max_{\pi} V^{\pi}(s)$. From the Bellman equations (2.4) and (2.5)it hence follows that any sub-sequence of

an optimal policy's trajectory must also be optimal, leading to the Bellmann optimality conditions:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)\Big[R(s,a,s') + \gamma V^*(s')\Big] \tag{2.6}$$

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a)\Big[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')\Big]. \tag{2.7}$$

Bellman equations (2.4) and (2.5) are the basis of *policy evaluation* algorithms, which estimate the value function of a given policy π. They can be combined with a *policy improvement* step, in which an improved policy $\pi'(s)$ can be found via $\pi'(s) = \arg\max_a Q^\pi(s,a)$. Iteratively applying policy evaluation and improvement leads to algorithms such as *policy iteration* and *value iteration*, which converge to the optimal policy and its value function. Value iteration, which combines the two steps into one, can be also thought of as a direct application of Bellman optimality conditions (2.6) and (2.7).

In the context of reinforcement learning, Bellman equations are used to update value function estimates given samples of transitions, or to derive the loss functions for training approximate value functions using deep learning.

2.2 Reinforcement Learning

Reinforcement Learning (RL) methods solve MDPs by learning from the interaction with the environment. Unlike planning methods (such as policy and value iteration), RL methods do not assume knowledge of the environment dynamics (i.e., $P_o(s)$, $P(s'|s,a)$ and $R(s,a,s')$), replacing the exact calculations of expectations with samples of transitions and rewards. We start this section by introducing the key concepts and then cover two main families of RL algorithms: temporal difference learning and policy gradients.

Credit Assignment Problem. One of the fundamental challenges is that of determining which actions are responsible for given outcomes. This problem is particularly acute in environments with sparse or delayed rewards, making it hard to correlate specific actions with their long-term outcomes.

Exploration. Refers to the act of trying new strategies to discover more about the environment or in an attempt to find better solutions. The challenge often is to balance exploration with *exploitation*, which refers to the selection of the best-known strategies. Too much exploration can lead to slow learning, while too little can result in agents that are stuck in sub-optimal strategies.

On-policy vs Off-policy. On-policy algorithms learn the value of a policy using the data generated by that policy. In contrast, off-policy algorithms may use data from a different policy. The policy whose value is being learned is referred to as the *target* policy, whereas the policy used to generate data is referred to as the *behavioral* policy.

Online vs Offline. The agent can be trained offline by using previously stored data, i.e., without real-time environment interaction, or on-line, by repeatedly interacting with the environment throughout training.

Model-based vs Model-free. Model-based algorithms incorporate learned models of environment dynamics, using the interaction data to also approximate P_0, P or R. Model-free algorithms learn only values and policies.

Function Approximation. In practice, value functions and policies are approximated with functions whose parameters are learned from data. Among other benefits, function approximation improves scalability, enables generalization to unseen state-action pairs, and enables the handling of continuous state-action spaces. In deep reinforcement learning, those functions are deep neural networks.

2.2.1 Temporal-difference Learning

Temporal-difference (TD) learning refers to the class of methods that rely on bootstrapping from the current estimates of value functions. They combine dynamic programming (i.e., iterative update rules based on policy evaluation or value iteration) with Monte Carlo estimation (i.e., the replacement of expectation

calculations with sampled interactions). In its simplest form, TD update of the value estimate of policy π is based on the Bellman equation (2.4):

$$V(s_t) \leftarrow V(s_t) + \lambda\Big[r_t + \gamma V(s_{t+1}) - V(s_t)\Big], \tag{2.8}$$

where the quantity $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is referred to as the *TD error* at time-step t and quantity $\hat{V}_{TD}(s) = r_t + \gamma V(s_{t+1})$ is referred to as the *target*, λ is the learning rate, and the transitions used in the update (s_t, a_t, r_t, s_{t+1}) are collected using π. The incremental updates with (2.8) have the effect of minimizing the TD error.

TD methods are guaranteed convergence in the limit where each state-action pair is visited infinitely often during learning and the learning rate λ satisfies the Robbins-Monro conditions (Sutton, 1988). The condition of sufficient state-action visitation is typically handled by using ϵ-greedy exploration strategy: with probability $1 - \epsilon$ the agent selects the action with the highest estimated value $a = \arg\max_a Q(s, a)$, and with probability ϵ, the agent selects a random action. Consequently, there is always a nonzero probability of selecting any action, satisfying the condition for visiting all state-action pairs infinitely often.

Sarsa. SARSA (State-Action-Reward-State-Action) is an on-policy algorithm with a form akin to policy iteration. The policy evaluation step uses the following TD update, with the target based on (2.5):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \lambda\Big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big], \tag{2.9}$$

where the transitions used in the update $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ are collected using the target policy. Note the TD error here is $\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$.

Q-Learning. Q-learning is an off-policy TD control algorithm that directly learns the optimal policy akin to value iteration. It can utilize transitions from behavioral policies that do not match the target policy, however, a large discrepancy in the

visitation of state-action pairs of the target and behavioral policy can result in poor sample efficiency. The target in Q-learning update is based on (2.7):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \lambda \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{2.10}$$

where the transitions (s_t, a_t, r_t, s_{t+1}) are collected using the behavioral policy and TD error has the following form $\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$.

TD Learning with Function Approximation

Both SARSA and Q-learning can be extended with function approximation to handle large or continuous state spaces where maintaining a table of Q-values for each state-action pair as in traditional TD learning is impractical or impossible. Instead, the objective becomes finding the parameters θ of a learned Q-function $Q(s, a; \theta)$ (or in a different notation $Q_\theta(s, a)$) that minimize the expected squared TD error:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\delta_t^2\right], \tag{2.11}$$

where in the case of Q-learning with function approximation the TD error is given by $\delta_t = r_t + \gamma \max_a Q(s_{t+1}, a; \theta) - Q(s_t, a_t; \theta)$ and in the case of SARSA with function approximation the TD error is given by $\delta_t = r_t + \gamma Q(s_{t+1}, a_{t+1}; \theta) - Q(s_t, a_t; \theta)$.

The Deep Q-Network (DQN) (Mnih et al., 2015) utilizes a neural network to approximate the Q-value function. The loss uses Q-learning TD error, and it is optimized with stochastic gradient descent (SGD). DQN stores the agent's experiences (s_t, a_t, r_t, s_{t+1}) from multiple episodes in a data set called the *experience replay buffer* $\mathcal{B}$. During training, the expectation in (2.11) is estimated with a sum over transitions sampled from the replay buffer. Random sampling of batches of experience breaks the temporal correlations in the sequences of experience tuples and leads to more stable and efficient training. This results in the following update rule for θ:

$$\theta \leftarrow \theta + \lambda \sum_{\substack{(s_t, a_t, r_t, s_{t+1}) \\ \in \mathcal{B}'}} \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) - Q(s_t, a_t; \theta) \right] \nabla_\theta Q(s_t, a_t; \theta), \tag{2.12}$$

where $\nabla_\theta Q(s_t, a_t; \theta)$ is the gradient of Q-value function at state-action pair (s_t, a_t) with respect to θ, $\mathcal{B}' \subset \mathcal{B}$ is a set of randomly sampled transitions, and θ^- are the parameters of a target network which is used specifically for target estimation. Target network has the same architecture as the Q-value function, but with only periodically updated parameters.

In addition to the introduction of the experience replay buffer and the use of target network, Mnih et al. (2015) stabilize the training further by clipping both rewards and TD errors to $[-1, 1]$ interval. Both measures have the effect of reducing large variations in reward scale or gradient size respectively. The combination of these innovations allowed DQN to learn successful policies in a high-dimensional visual domains such as Atari video games, which was a significant milestone in the development of deep reinforcement learning.

2.2.2 Policy Gradients

In contrast to TD learning, which learns a value function and derives a policy based on this value function, policy gradient (PG) methods parameterize and directly learn the policy (Sutton et al., 1999a). The objective in policy gradient methods is to maximize the expected return by directly adjusting the parameters θ of the policy $\pi_\theta(a|s)$. One of the possible formulation of such loss $\mathcal{J}(\theta)$ is given by:

$$\mathcal{J}(\theta) = -\mathbb{E}_{\rho^\pi, \pi_\theta}\Big[Q^\pi(s, a)\Big], \tag{2.13}$$

where $\rho^\pi(s)$ represents the state distribution under policy π_θ and $Q^\pi(s, a)$ is the Q-function under π_θ.

Policy gradient theorem provides a foundation of the learning algorithm: it states that the gradient of the expected return with respect to the policy parameters does not involve derivative through the state distribution and hence does not require knowledge of the environment dynamics. Through the policy gradient theorem, the gradient of 2.13 has the following form:

$$\nabla_\theta \mathcal{J}(\theta) = -\mathbb{E}_{\rho^\pi, \pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)\Big]. \tag{2.14}$$

The equation (2.14) indicates that we can improve the policy by adjusting the parameters in the direction of the gradient of $\log \pi_\theta(a|s)$ weighted by the Q-function which evaluates the goodness of the action taken. To reduce high variance in gradient estimates and improve learning stability, a common practice is to incorporate a *baseline* – a function of state $b(s)$ which can be shown to not affect the gradient (2.14) in expectation and the most common choice of baseline is the state-value function $V^\pi(s)$. The baseline-subtracted term $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is referred to as the *advantage*, indicating whether the action's outcome was better or worse than the baseline expectation for that state. With the state-value baseline, the policy gradient has the following form:

$$\nabla_\theta \mathcal{J}(\theta) = -\mathbb{E}_{\rho^\pi, \pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s)\Big(Q^\pi(s,a) - V^\pi(s)\Big)\Big]. \tag{2.15}$$

Like other RL methods, PGs also require exploration to avoid getting stuck in suboptimal policies. A common approach in DRL is entropy regularization – the standard objective function 2.13 is augmented with a term $\alpha H(\theta)$ that encourages policy entropy:

$$\alpha H(\theta) = -\alpha\ \mathbb{E}_{\pi_\theta}\Big[\log \pi_\theta(a|s)\Big], \tag{2.16}$$

where the parameter α manages the trade-off between exploration and exploitation. The addition of the entropy term to the objective incentivizes the policy to be less deterministic, effectively encouraging a more uniform probability distribution over actions.

Actor-Critic

In the discussion of policy gradients so far, we did not specify how to estimate the advantage in (2.15). Actor-Critic (AC) methods, which integrate the advantages of policy gradient methods with the benefits of TD learning, offer one approach to this problem (Konda & Tsitsiklis, 1999). AC algorithms consist of two separately parametrized and concurrently trained functions: policy (*actor*) and value function (*critic*). The actor $\pi_\theta(a|s)$ is trained using policy gradients with the critic providing

the estimate of the advantage function, while the critic $V_\phi(s)$ is trained using TD learning methods. In a general case, the critic can be either (or both) state-value or action-value function. One of the ways to obtain an unbiased estimate of advantage from the critic is as follows:

$$A^\pi(s_t, a_t; \phi) = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{2.17}$$

where the transitions (s_t, a_t, r_t, s_{t+1}) are collected under π_θ and $V_\phi(s_t)$ is the value function learned from the same data as π_θ. By reducing the variance in the gradient estimatation, AC methods tend to be more data-efficient compared to traditional policy gradient methods.

Soft Actor-Critic. Soft Actor-Critic (SAC) specifically addresses the exploration-exploitation trade-off through a policy that is trained to maximize not just the expected return, but also the entropy of the policy (Haarnoja et al., 2018). Note that this is similar to actor-critic with entropy regularization, however, SAC is derived from the maximum entropy reinforcement learning framework (Ziebart, 2010) and the entropy is incorporated more thoroughly.

The updates in SAC consist of learning three separate neural networks: two Q-value networks and one policy network. Similar to double Q-learning (van Hasselt et al., 2016), the use of two Q-value functions counteracts the maximization bias – a phenomena where the Q-learning algorithm systematically overestimates action-values due to the use of max operator. The loss for a single Q-value network Q_{ϕ_i} is given by:

$$\mathcal{L}(\phi_i) = \sum_{\substack{(s_t, a_t, r_t, s_{t+1}) \\ \in \mathcal{B}'}} \left[\left(r_t + \gamma \Big(\min_{j=1,2} Q_{\phi_j^-}(s_{t+1}, a') - \alpha \log \pi_\theta(a'|s_{t+1}) \Big) - Q_{\phi_i}(s_t, a_t) \right)^2 \right], \tag{2.18}$$

where $Q_{\phi_j^-}$ represents the target network of the j-th Q-value function, $\mathcal{B}'$ is the set of sampled experiences, and a' is sampled from the current policy $\pi_\theta(\cdot|s_{t+1})$. The policy gradient also incorporates an entropy term, effectively directly managing the trade-off between expected return and policy entropy:

$$\mathcal{J}(\theta) = -\mathbb{E}_{\rho_\pi, \pi}\Big[Q_\phi(s, a) - \alpha H(\theta) \Big], \tag{2.19}$$

where α is the temperature parameter that determines the significance of the entropy component in the overall objective and ρ^{π} is the state distribution under π_θ.

SAC uses the *reparameterization trick* when estimating the policy gradient – a technique for enabling backpropagation through a stochastic node in a computational graph by reparametrizing the stochastic node as a deterministic function of input noise variable. In this case, the actions $a \sim \pi_\theta(\cdot|s)$ can be expressed as:

$$a = f_\theta(\xi; s), \tag{2.20}$$

where ξ is a noise input sampled from some fixed distribution, typically $\mathcal{N}(0, 1)$. With this trick, we can express the policy loss as an expectation over noise instead of actions. Finally, by also utilizing the minimum of the two Q functions, the loss becomes:

$$\mathcal{J}(\theta) = -\mathbb{E}_{\rho_\pi, \mathcal{N}} \left[\min_{j=1,2} Q_{\phi_j}\Big(s, f_\theta(\xi; s)\Big) - \alpha \log\ \pi_\theta\Big(f_\theta(\xi; s)|s\Big) \right]. \tag{2.21}$$

Proximal Policy Optimization Proximal Policy Optimization (PPO) is designed to address challenges associated with the stability and sample efficiency of PG methods (Schulman et al., 2017). Traditional policy gradient methods can suffer from large policy updates that destabilize training. An alternative is Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), which while being more sample efficient and stable by constraining the updates to a specific trust region, is prohibitively computationally expensive because it requires second-order optimization. PPO has a similar effect to TRPO, however, it replaces the expensive second-order optimization with a much less computationally expensive clipped surrogate objective for the policy:

$$\mathcal{J}_{\text{CLIP}}(\theta) = -\mathbb{E}_{\rho_\pi, \pi_\theta} \left[\min \left(\rho(\theta)\hat{A},\ \text{clip}\Big(\rho(\theta), 1-\beta, 1+\beta\Big)\hat{A} \right) \right], \tag{2.22}$$

where $\rho(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ is the probability ratio of new to the old policy, $\hat{A}$ is the estimate of advantage (such as 2.17) under $\pi_{\theta_{old}}$, and β is a small parameter defining the clipping range. The objective $\mathcal{J}_{\text{CLIP}}(\theta)$ takes the minimum of the unclipped

and clipped surrogate objectives, effectively penalizing changes to the policy that move $\rho(\theta)$ outside of the $[1-\beta, 1+\beta]$ interval.

As an on-policy algorithm, PPO alternates between collection of data through interaction with the environment and optimization of the clipped surrogate objective function via SGD and doing multiple passes through the data.

2.2.3 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) aims to address the challenges in long-horizon SDM with large state and action spaces. Such problems pose difficulties for standard RL methods due to the exponentially large search space and issues with sparse rewards and credit assignments. The key idea of HRL is *temporal abstraction* - instead of taking only low-level (primitive) actions as in standard RL, the HRL agent has a hierarchical policy (controller) that can invoke high-level skills or subtasks that persist over extended periods of time before terminating (Barto & Mahadevan, 2003; Pateria et al., 2021). Such form of temporal abstraction can effectively shorten the effective planning horizon and enable transfer between tasks that share structural similarities.

In this work, we only consider the HRL methods with two levels of abstraction: a high-level policy $\pi_{hi}(z|s)$ that selects skills, goals or instructions $z \in Z$; and a lower-level policy $\pi_{lo}(a|s,z)$ (or alternatively a set of separately parametrized policies $\{\pi_{lo}^{(z)}(a|s)\}$) that selects primitive actions $a \in \mathcal{A}$. Skills z chosen by π_{hi} persist for an extended duration by executing π_{lo} repeatedly, before terminating according to either a termination condition given by function $\beta_z(s)$ or after some predefined fixed number of steps. The π_{lo} meanwhile executes on a particular skill or tries to achieve the subgoals defined by z.

The entire system of higher and lower-level policies can be trained to maximize the extrinsic reward (i.e., the rewards originating from the environment) (Bacon et al., 2017). Alternatively, lower-level policies can be provided with also intrinsic rewards (i.e., rewards constructed to help with training, but which are not used to evaluate the policy performance) for reaching sub-goals related to z (Kulkarni

et al., 2016). In practice, however, both approaches can suffer from problems. The end-to-end training tends to suffer from *hierarchy collapse* (Bacon et al., 2017) – the phenomenon in which higher-level policy learns to always select the same action z or the lower-level policy conditional on z learns to ignore it – effectively resulting in a flat, non-hierarchical policy. On the other hand, constructing an intrinsic reward for reaching sub-goals can require domain expertise or manual coding of reward functions.

Another common approach is to construct lower-level policies or skills by hand, and then train only a higher-level policy over such skills (with or without fine-tuning of the lower-level skills along the way). The lower-level skills can be constructed by selecting (and constructing) a set of tasks M_z, where each skill z corresponds to a solution of M_z.

Variational Option Discovery

Variational option discovery (VOD) methods (Gregor et al., 2016; Eysenbach et al., 2019; Achiam et al., 2018) are a recently proposed family of techniques for unsupervised skill discovery. A skill π_z here is defined as a policy $\pi(a|s,z)$ conditioned on a discrete latent variable z. VOD algorithms aim to discover a set of diverse skills such that each skill z can be uniquely identified from some statistic ξ_τ derived from the trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$ generated by $\pi_\theta(a|s,z)$. Typical choices for ξ_τ are a sub-sequences of visited states $\{s_0, s_k, s_{2k}...\}$ as in Achiam et al. (2018), individual states s_t as in Eysenbach et al. (2019) or terminal states s_T as in Gregor et al. (2016).

Formally, the skills are trained to maximize the mutual information (MI) between z and the statistic ξ_τ, $I(z;\xi_\tau)$, sometimes including maximization of the entropy $H[\pi_\theta(a|s,z)]$ to help ensure good exploration.

Due to the symmetry of MI, two decompositions are generally employed:

$$I(z;\xi_\tau) = H(z) - H(z|\xi_\tau) \tag{2.23}$$

$$= H(\xi_\tau) - H(\xi_\tau|z), \tag{2.24}$$

where (2.23) is called the *reverse* MI and (2.24) the *forward* MI (Gregor et al., 2016; Campos et al., 2020). Optimizing either form directly is generally intractable.

Fortunately, the reverse form admits maximization of a variational lower bound instead, resulting in the entropy-regularised objective:

$$\max_{\theta,\phi} \mathbb{E}_{z\sim P(z)}\Bigg[\mathbb{E}_{\tau\sim\pi_z,T,P_0}\Big[\log q_\phi(z|\xi_\tau) + \alpha H\Big(\pi_\theta(a|s,z)\Big)\Big] - \log P(z)\Bigg]. \tag{2.25}$$

By contrast, the forward objective can only be bounded approximately, due to the intractability of both $\log P(\xi_\tau|z)$ and $\log P(\xi_\tau)$, which have opposite signs. This approximation (with entropy regularisation) is given by:

$$\max_{\theta,\phi} \mathbb{E}_{z\sim P(z)}\Bigg[\mathbb{E}_{\tau\sim\pi_z,T,P_0}\Big[\log q_\phi(\xi_\tau|z) - \log \sum_{z'} q_\phi(\xi_\tau|z')P(z') + \alpha H\Big(\pi_\theta(a|s,z)\Big)\Big]\Bigg]. \tag{2.26}$$

Despite this, the forward objective has been used effectively (Sharma et al., 2020; Campos et al., 2020), indicating that the applied variational models are able to closely fit the true distribution.

In both cases, the latent distribution $P(z)$ is usually not learned and set to be uniform to ensure that all skills are trained equally. The training procedure consists of two simultaneous optimization processes. The variational model, referred to as the *discriminator*, is trained to minimize the negative log-likelihood loss $-\log q_\phi$; while a skill-conditional policy maximizes the corresponding lower bound in either (2.25) or (2.26). In other words, for the reverse objective, the discriminator is trained to predict skills from the trajectories, while for the forward objective, the discriminator is trained to predict the distribution of states visited by the skill. In both cases, policies are rewarded for producing trajectories on which the discriminator performs well. In practice, skills learned via forward MI tend to be unimodal, since each skill is specified by a single state, while reverse mode skills are often multi-modal (Campos et al., 2020).

2.3 Imitation Learning

MDP framework is also used in Imitation Learning (IL), where the objective is to train an agent from observed expert demonstrations rather than through interaction

with the environment. The primary goal of IL is to replicate the demonstrator's performance accurately and efficiently, often with far fewer interactions with the environment than would be required when learning from scratch.

Imitation learning approaches can be broadly categorized into two types: Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL). BC approaches rely essentially just supervised learning and involve only learning the mapping from observed states to actions. In contrast, IRL infers an underlying reward function that explains the expert's decisions and uses it to derive a policy. While both methods learn from expert demonstrations, they fundamentally differ in their interpretation of the expert behavior and the learning process.

In addition to BC and IRL, modern techniques have emerged to tackle some of the inherent problems in imitation learning. Interactive imitation learning, such as DAgger (Ross et al., 2011), assume access to the expert during training. During the iterative learning process, the expert provides provides correct actions for states visited by the learner. Adversarial imitation learning, exemplified by GAIL (Ho & Ermon, 2016), frames the imitation learning problem as a game between the imitator (policy) and a discriminator. The imitator is rewarded for generating actions indistinguishable from the expert's actions, circumventing the need to model the reward function explicitly as in IRL. Imitation from observation goes even further, learning purely from state information without the need for explicit action demonstrations (Torabi et al., 2019).

2.3.1 Behavioral Cloning

BC is conceptually straightforward: it is a form of supervised learning where the labeled data (state-action pairs generated by the demonstrator) is used to learn a policy. The algorithm structure typically involves collecting a dataset of demonstrations $\tau = \{(s_t, a_t)\}_{t=0}^{T}$ from an expert, and then using this dataset, training a function approximator $\pi_\theta(s)$, such as a neural network, to predict the expert's actions given a state s.

Loss functions in BC are typically a mean square error when the action space is continuous and cross-entropy when the action space is discrete. The optimization process involves adjusting the parameters of the policy to minimize the predicted actions' deviation from the expert actions:

$$\mathcal{L}(\theta) = \sum_{(s_t,a_t)\in\tau} f\Big(\pi_\theta(s_t), a_t\Big), \tag{2.27}$$

where $f(\cdot)$ is the suitable loss function (i.e., mean square error or cross-entropy).

One significant assumption in BC is that the demonstration data comprises independent and identically distributed (i.i.d.) samples. In a sequential decision-making context, however, this assumption does not hold because consecutive states are often correlated. As the learning agent deviates from the states seen during training (due to non-i.i.d. nature of sequential data), it can incorrectly generalize its behavior to these unseen states, which in turn leads to further errors. Such problems motivated the development of new forms of imitation learning which, for example, incorporate the querying of an expert during training (such as DAgger), or training with reinforcement learning (such as GAIL).

2.3.2 Inverse Reinforcement Learning

The IRL framework seeks to explain the experts' demonstrations through an underlying reward function, which is not directly observed or known. The idea is that if the expert is optimizing a reward, one can infer what that reward must be by observing the expert's behavior. The training objective for IRL is to find a reward function $R(s, a)$ such that the expert policy is the best policy under this reward function, indicative by the following optimization problem:

$$\max_R \sum_{(s_i,a_i)\in\tau} R(s_i, a_i) - \log Z(R), \tag{2.28}$$

where $Z(R)$ is the partition function that normalizes the probability distribution over all possible actions in all states. Once inferred, this reward function can then be used to derive an optimal policy, potentially with any RL algorithm. Classical IRL approaches often involve solving a bi-level optimization problem

where one must perform reinforcement learning as an inner loop to determine the best policy for a given reward function and then adjust the reward function to better explain the observed behavior.

A significant contribution to IRL research was the proposal of the Maximum Entropy IRL framework (Ziebart et al., 2008), which resolves ambiguities in the reward function by favouring solutions that explain the expert's actions while being maximally uncertain about unobserved actions.

2.4 Meta-Gradients

The successful training of an RL agent is significantly dependent on the appropriate selection of hyper-parameters such as the discount factor, learning rate, or exploration rate (Henderson et al., 2018; Andrychowicz et al., 2020). Conventionally, these hyper-parameters are selected via cumbersome manual search, or through computationally expensive grid or random search methods (Bergstra & Bengio, 2012).

The meta-gradient literature (Xu et al., 2020; Zahavy et al., 2020) makes a distinction between the parameters θ (e.g., weights of the policy and value networks) and the meta-parameters η (e.g., learning rates, discounts, weights of regularization losses). The parameters θ are trained by minimizing an inner loss function $\mathcal{L}_{\eta}^{(\text{inner})}(\theta, \mathcal{D})$ parameterized by the meta-parameters η, whereas the meta-parameters η are trained to minimize the outer loss $\mathcal{L}^{(\text{outer})}(\eta, \mathcal{D})$, where in both cases $\mathcal{D}$ refers to the rollout data used to estimate the loss. The parameters and meta-parameters are generally optimized at two different time-scales; here we denote by i the inner loop and by k the outer loop iteration step.

Inner and Outer Objectives

Most meta-gradients are applied on entropy-regularized AC algorithms (Xu et al., 2018; Zahavy et al., 2020), where the inner loss objective $\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D})$ consists of the following three terms (policy, value and entropy loss respectively):

$$\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D}) = \alpha_{\pi}\mathcal{L}_{\pi}(\theta, \mathcal{D}) + \lambda_{v}\mathcal{L}_{v}(\theta, \mathcal{D}) + \alpha_{\text{ent}}\mathcal{L}_{\text{ent}}(\theta, \mathcal{D}). \tag{2.29}$$

The hyper-parameters required for computation of each of the three terms (for example, discount γ and bootstrapping parameters λ in n-step returns) or the weights of individual losses, can become tuneable meta-parameters.

The outer loss objective generally consists of the same three terms, weighted by hyper-parameters $\alpha_{\pi}^{(\text{outer})}, \alpha_{v}^{(\text{outer})}, \alpha_{ent}^{(\text{outer})}$:

$$\mathcal{L}^{(\text{outer})}(\eta, \mathcal{D}) = \alpha_{\pi}^{(\text{outer})} \mathcal{L}_{\pi}\Big(\theta(\eta), \mathcal{D}\Big) + \alpha_{v}^{(\text{outer})} \mathcal{L}_{v}\Big(\theta(\eta), \mathcal{D}\Big) + \alpha_{\text{ent}}^{(\text{outer})} \mathcal{L}_{\text{ent}}\Big(\theta(\eta), \mathcal{D}\Big), \tag{2.30}$$

where the data $\mathcal{D}$ used for estimating the outer loss could be the same as in inner loss, or come from a separate set of rollouts. The outer loss objective depends on the meta-parameters η through their influence on the learned parameters θ. This dependence is given by the update rule: $\theta_{i+1} = f(\theta_i, \eta_k, \mathcal{D}_i)$, where $\mathcal{D}_i$ refers to the data used to compute the losses at i-th iteration. For example, for vanilla SGD, $\theta_{i+1} = \theta_i - \nabla_{\theta} \mathcal{L}_{\eta}^{(\text{inner})}(\theta_i, \mathcal{D}_i)$, which is parameterized by η_k through $\mathcal{L}_{\eta}^{(\text{inner})}(\theta_i, \mathcal{D}_i)$. Since backpropagating through the entire history of updates is too computationally demanding and may not provide a good gradient estimate due to non-stationaritity intrinsic to RL, the meta-gradients are truncated to the last K inner loop updates, which is referred to as *meta rollout length*. The outer losses can be computed on a new rollout data $\mathcal{D}_k$ or the rollout data used in the inner loop $\mathcal{D}_i$.

2.4.1 Bootstrapped Meta-Gradients

Propagation the meta-gradient through only the last K steps results in a short-horizon bias: the meta-learning fails to account for the longer-term changes to the optimization process (Wu et al., 2018). When K is large enough to account for longer-term changes, meta-optimization can result in exploding gradients (Metz et al., 2019). To address there problems, Flennerhag et al. (2022) propose the improved outer objective: the meta-parameters are trained to minimize a distance to a target which has been bootstrapped from the meta-learner. This method is referred to as Bootstrapped Meta-Gradients (BMG). In the best studied version of BMG, the meta-gradients are propagated through the last K updates, while minimizing the KL-divergence between the policy parametrized with θ_K and a

bootstrap target $\pi_{\hat{\theta}}$ obtained by optimizing the policy for another $L-1$ steps under the meta-learned update rule:

$$\mathcal{L}_{\text{BMG}}^{(\text{outer})} = KL(\pi_{\hat{\theta}}||\pi_{\theta_K}), \tag{2.31}$$

where the hyperparameter L is referred to as bootstrap target length. The use of target which is $L-1$ steps ahead (without increasing the number of steps the meta-gradients are backpropagating through) reduces the myopia of standard MG objectives, while the use of KL divergence reduces ill-conditioning of outer loop objective. The resulting adaptations of meta-parameters encourage reaching the target policy in a smaller number of inner optimization steps.

2.5 Transfer in Natural Language Processing

Transfer learning has become critical in state-of-the-art practices across nearly all domains of natural language processing (NLP). With smart leveraging of pretrained representations and model parameters obtained by training on large corpora of unlabeled textual data, transfer learning approaches deliver substantial performance gains over training on target tasks in isolation.

One of the most powerful transfer methods in NLP involves transferring word embeddings, i.e., fixed-size continuous representations of words or subwords. Pretrained word embeddings like word2vecor and GloVe (Mikolov et al., 2013; Pennington et al., 2014) capture syntactic and semantic information that can be transferred to new tasks. The pretrained word vectors can be used to initialize the input embedding layers of neural networks in downstream tasks (Tenney et al., 2019; Peters et al., 2018b), or outside of NLP, to shape representations in image classification and enable prediction of labels not seen during training (Socher et al., 2013; Frome et al., 2013).

In addition to word embeddings, in recent years, transfer from pretrained language models has become very popular. Models like BERT, GPT, Llama (Devlin et al., 2018; Radford et al., 2019; Touvron et al., 2023a) and others learn complex language representations and capabilities by learning to predict a word from its context. The rich knowledge encapsulated in these powerful models can then be

transferred to downstream tasks. There are a few common approaches to transfer learning with language models. One is fine-tuning, where the language model is adapted to the downstream task by appending task-specific output layers (e.g., a classification head as in sentiment analysis or multiple choice question answering) and fine-tuning the entire model to the target data (Howard & Ruder, 2018; Minaee et al., 2021). Another approach is feature extraction, where instead of fine-tuning, the pretrained model's hidden activations are used as input features to train a smaller downstream model (Peters et al., 2018a; Zhu et al., 2019; Zhang et al., 2020) or as inputs to simple clustering algorithms (Eklund & Forsman, 2022). Finally, prompt programming reformulates tasks into cloze-style prompts for the language model (Petroni et al., 2019; Liu et al., 2023).

2.5.1 Word Embeddings

Word embeddings represent words or phrases as vectors of real numbers in a high-dimensional space. These embeddings capture semantic and syntactic properties of words, such that, for example, words with similar meaning tend to be closer in the representation space. There are two main types of word embeddings: non-contextual and contextual embeddings. In context-free embeddings, a word has the same representation regardless of its context. That is, the word 'bark' will have the same embedding in 'tree bark' and 'dogs bark'. On the other hand, contextual embeddings generate representations that take into account the context in which a word appears. This means that the same word can have different embeddings based on its surrounding words. Both classes of models are motivated by Firth's distributional hypothesis ("You shall know a word by the company it keeps") (Firth, 1957), which suggests that the learned vector representation of a word like 'chihuahua' should be similar to 'doberman' if the corresponding words appear in similar contexts, e.g., if they can both be found around other words like 'pet' or 'bark'.

Some of the examples of context-free embeddings are word2vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014). Word2vec (Mikolov et al., 2013) operates using either of two architectures, Continuous Bag of Words (CBOW) or Skip-Gram,

to learn word associations from a large corpus of text by predicting a target word from the surrounding context words (CBOW) or predicting context words from a target word (Skip-Gram). Both models are shallow neural networks. One of the fascinating properties of word2vec representations is the emergence of meaningful semantic relationships in the algebraic manipulations of word vectors. For example, by subtracting the vector of 'man' from the vector of 'king' with the addition of the vector of 'woman', the result often comes very close to the vector for 'queen'. Similar relationships have been found for country-capital and tense-past tense relations.

GloVe (Pennington et al., 2014) obtains the vector representations of words from the global co-occurrence statistics of words within a corpus. The training objective of GloVe is to minimize the difference between the dot product of word vector pairs $(w_i, \tilde{w}_j)$ and the logarithm of their co-occurrence counts X_{ij} (i.e., the number of times word i occurs next to j), adjusted with a special weighting function $f(X_{ij})$ to ensures the model does not overweight the importance of very frequent word co-occurrences. This can be expressed with the following objective:

$$\mathcal{L}(W, \tilde{W}) = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}j - \log X_{ij} \right)^2, \tag{2.32}$$

where V is the size of the vocabulary and $(b_i, \tilde{b}_j)$ are the scalar bias terms for words i and j. The model learns two sets of embeddings, $W = \{w_i\}$ and $\tilde{W} = \{\tilde{w}_i\}$, with the final embedding for i-th word set as $w_i + \tilde{w}_i$. Similar to word2vec, GloVe embeddings also demonstrate semantic relationships in the algebraic manipulations of word vectors.

Contextual word representations are obtained by taking hidden states of the pre-trained language model, which can provide embeddings of entire sentences and capture more complex relationships between words. We introduce language models in the next section.

2.5.2 Large Language Models

Large language models (LLMs) represent a significant advancement in the field of natural language processing (NLP). These models are designed to generate

human language, in the process achieving language processing capabilities previously thought to be exclusive to human cognition. The underlying premise is that the structure of language, the ability to reason and the knowledge of the world that is captured in the vast textual corpora can be successfully learned by using large enough training datasets and numbers of model parameters. LLMs have shown superiority in a wide array of NLP tasks, from commonsense reasoning and summarization to question answering. With large enough training datasets and model parameters, LLMs out-compete specialized models even when evaluated in zero or few-shot setting, i.e., without any fine-tuning on the downstream task with the option to provide a small number of correct answers as examples.

Tokenization

A fundamental element in LLMs is the concept of a *token*, which represents the basic unit of text, such as a word or part of a word. In the input and output of a LLM, each token is represented with a one-hot vector, with dimensions determined by the total size of the vocabulary.

The process of breaking down text into tokens, referred to as tokenization, transforms raw language data into a format that can be processed by the model. The choice of tokenization is important as it impacts the size of the model's vocabulary, and can influence the model's ability to learn tasks and generalize to new, unseen text. For example, it has been shown that the method of tokenizing numbers can affect the performance of models on tasks requiring numeracy (Wallace et al., 2019; Golkar et al., 2023).

Some of the most common tokenization schemes are Byte Pair Encoding (BPE) (Sennrich et al., 2015), WordPiece (Wu et al., 2016) and SentencePiece (Kudo & Richardson, 2018), each of which provides a way of dealing with out-of-vocabulary words. BPE, for example, merges the most frequent character pairings and iteratively builds a vocabulary until reaching a specified vocabulary size. WordPiece and SentencePiece refine the BPE algorithm to better handle rare and out-of-vocabulary words, as well as to handle diverse languages.

Objective

LLMs typically employ one of two main training objectives: auto-regressive and masked token prediction. Auto-regressive models, such as GPT (Radford et al., 2018), are trained to predict the next token in a sequence given the preceding tokens. In other words, auto-regressive model assumes that each word is determined by its past and not by its future. This can be expressed with the following cross-entropy loss:

$$\mathcal{L}_{\text{auto-reg}}(\theta) = -\sum_{i} \log p(x_i|x_{<i};\theta), \tag{2.33}$$

where x_i is the target token, $x_{<i}$ is the sequence before the i-th token. For masked token models like BERT (Devlin et al., 2018), a randomly set of tokens in the input sequence are masked, and the model is trained to predict these masked tokens, thereby learning prediction from bidirectional context. The associated loss function can be written as:

$$\mathcal{L}_{\text{masked}}(\theta) = -\sum_{i \in M} \log p(x_i|x_{\neg M};\theta), \tag{2.34}$$

where M is the set of masked positions, and $x_{\neg M}$ represents the remaining, non-masked tokens in the sequence.

Architecture

The initial work on language modeling with deep neural networks, such as ELMo (Peters et al., 2018a) and ULMFit (Houlsby et al., 2019), relied on the LSTM architecture (Hochreiter & Schmidhuber, 1997). However, since their introduction in Vaswani et al. (2017b), transformers emerged as the predominant architecture for LLMs. This is due to their ability to handle very long sequences of data (via the use of self-attention mechanisms) and effective parallelization (via position-wise feed-forward networks), with skip connections enabling propagation of gradients even in very deep neural networks.

Each layer of a transformer consists of two sub-layers: a multiple-head self-attention and position-wise fully connected feed-forward network, with skip connections allowing bypassing of each sub-layer. The input to each layer is a sequence of

vectors, with each vector corresponding to an embedding of the corresponding input token in the sequence (while also containing positional information to maintain the order of tokens in a sequence). The transformations of each layer do not change the length of the sequence or the dimensionality of embedding, instead, they imbue the embeddings with increasingly rich information about the complex relationships between the tokens.

The self-attention mechanism is the most important and computationally expensive element of the transformer layer. Each attention head computes the similarity of (specific to that head) representations between all pairs of tokens in the sequence, then uses these similarities to weigh the importance of different parts of the input data differently and capture contextual relations regardless of distance within the text. The longest sequence of tokens that a given model is trained to process through self-attention is referred to as the context window.

The differences between auto-regressive and masked token model objectives are also reflected in the application of self-attention. Importantly, auto-regressive models like GPT Radford et al. (2018) also utilize a causal (or unidirectional) masking in the self-attention mechanism: in the computation of weights, all future tokens are masked, ensuring that the prediction for a particular token only depends on previously generated tokens. Masked token models on the other hand utilize bidirectional attention, allowing each token to be conditioned on tokens from both the past and future context within a single layer.

The parameter count of a typical transformer model is primarily determined by the number of layers (depth), token embedding dimension and the size of feed-forward networks. Notably, however, the model parameter count is typically not determined by the context window.

Training Datasets

The data for training LLMs is sourced from a wide range of textual sources to encompass a broad spectrum of language usage, but mostly data collected in large-scale web scraping efforts like Common Crawl (Patel & Patel, 2020). Overall, the

data sources include: web-content (text and code from websites, blogs, news articles, and forums), books and literature (collections of books, both fiction and non-fiction), scientific articles, social media (Twitter and Reddit) and other specialized sources (Radford et al., 2019).

Given the vastness and the public nature of these sources, data filtering and pre-processing are needed to ensure the quality and relevance of the training data, as well as to minimize the amount of sensitive or personal data. The filtering process involves filtering out of duplicate and spam content, minimizing the amount of harmful, biased and sensitive content, and standardization of the data format (such as removal of html tags).

Emergent Properties

By increasing the parameter count of LLMs and the size of the training dataset, LLMs start exhibiting emergent capabilities that these models were not explicitly trained for, but that arise from the difficulty of the training objective, the richness and size of the training data and the architecture (Wei et al., 2022). While the early LSTM and transformer-based models had millions of parameters and were trained on billions of tokens (Peters et al., 2018a; Devlin et al., 2018), the modern LLMs have billions to trillions of parameters and are trained on datasets of trillions of tokens (Hoffmann et al., 2022; Chowdhery et al., 2022; Touvron et al., 2023b).

LLMs demonstrate an ability to perform tasks with minimal instruction or numbers of examples. Unlike traditional models that require extensive task-specific training data, LLMs can generalize from a few task-specific examples, where the task-specific examples are provided in the context window as a part of the query prompt. In some cases, LLMs can perform tasks without any task-specific examples, just interpreting the instruction or natural language descriptions of a task (Brown et al., 2020). This ability of LLMs to adapt their responses based on the context, but without changing the model weights, is referred to as *in-context learning*. It suggest an underlying understanding of language and concepts that goes beyond memorization of the training data.

Models trained on large datasets can perform simple arithmetic, solve word problems, explain jokes, and even demonstrate logical reasoning, despite not being explicitly trained for these tasks. They have also shown a capacity for creative generation, producing content that includes coherent and contextually appropriate stories, dialogue, poetry, and code. The most capable LLMs such as GPT-4 (OpenAI, 2023a) and Claude 2 (Anthropic, 2023) are even exceeding the performance of average humans on many of these tasks (Bubeck et al., 2023).

An important pattern observed in training increasingly larger LLMs is the manifestation of scaling laws. As the amount of computation (compute) dedicated to training of these models increases, their performance improves in a predictable manner. This trend holds across various tasks, and for various ways of distributing increases in compute (either through parameter count, training tokens, or batch size), with empirical works suggesting how to best allocate the compute for the largest performance gains (Kaplan et al., 2020; Hoffmann et al., 2022). These results are part of the motivation for large investments in this space, with the expectation of unlocking new abilities with each progressively larger, new generation of models.

3

Natural Language for Reinforcement Learning

Contents

3.1 Introduction

Languages, whether natural or formal, allow us to encode abstractions, generalize, to communicate plans, intentions, and requirements, both to other parties and to ourselves (Gopnik & Meltzoff, 1987). These are fundamentally desirable capabilities of artificial agents. However, agents trained with traditional approaches within dominant paradigms such as Reinforcement Learning (RL) and Imitation Learning (IL) typically lack such capabilities, and struggle to efficiently learn from interactions with rich and diverse environments. Throughout this Chapter, which is based on

Luketina et al. (2019), we argue for natural language as one of the key solutions to sequential decision making problems (*i.e.* those often approached with RL[1]). We survey recent work and tools that are beginning to make this shift possible and outline the next research steps.

Humans are able to learn quickly in new environments due to a rich set of commonsense priors about the world (Spelke & Kinzler, 2007), some of which are reflected in natural language (Shusterman et al., 2011; Tsividis et al., 2017). It is thus reasonable to question whether agents can learn not only classic interactions with the environment – that is, from rewards or demonstrations, but also from information encoded using language, in order to improve generalization and sample efficiency—especially when (a) learning is severely constrained by data efficiency due to limited or expensive environment interactions, and where (b) human priors that would help to solve the task are or can be expressed easily in natural language. Furthermore, many, if not most, real-world tasks require agents to process language by design, whether to enable interaction with humans or when using existing interfaces and knowledge bases. This suggests that the use of language in RL has both broad and important applications.

Information contained in both generic and task-specific large textual corpora may be highly valuable for decision making. Pre-training neural representations of words and sentences from large generic corpora has already shown great success in transferring syntactic and, to some extent, semantic information to downstream tasks in natural language understanding (Peters et al., 2018b; Goldberg, 2019; Tenney et al., 2019). Cross-domain transfer from self-supervision on generic language data might similarly help to initialize RL agents. Task-specific corpora like wikis or game manuals could be leveraged by machine reading techniques (Banko et al., 2007) to inform agents of valuable features and policies (Eisenstein et al., 2009; Branavan et al., 2012), or task-specific environmental dynamics and reward structures (Narasimhan et al., 2018; Bahdanau et al., 2019).

[1]We write RL for brevity and to focus on a general case, but our arguments are relevant for many sequential decision making approaches, including IL and planning.

Previous attempts at using language for RL tasks in these ways have mostly been limited to relatively small corpora (Janner et al., 2018), or synthetic language (Hermann et al., 2017). We argue that recent advances in representation learning (Peters et al., 2018a; Devlin et al., 2018; Radford et al., 2019) make it worth revisiting this research agenda with a much more ambitious scope. While the problem of grounding (*i.e.* learning the correspondence between language and environment features) remains a significant research challenge, past work has already shown that high-quality linguistic representations can assist cross-modal transfer outside the context of RL (*e.g.* using semantic relationships between labels to enable zero-shot transfer in image classification (Frome et al., 2013; Socher et al., 2013)).

In the rest of this Chapter (which is based on our survey paper (Luketina et al., 2019) and updated with more recent works), we review prior work, considering settings where interaction with language is necessary (§3.2.1) and where language can optionally be used to facilitate learning (§3.2.2). In the former category, we review instruction following, induction of reward from language, and environments with text in the action or observation space, all of which have language in the problem formulation itself. In the latter category, we review work that uses language to provide auxiliary rewards, facilitate RL by transfer from domain-specific textual resources, or shape structure learned models. We conclude by identifying what we believe are the most important challenges for integrating natural language in RL (§3.3). Inspired by gaps in the existing literature at the time, in Luketina et al. (2019) we advocated the development of new research environments utilizing domain knowledge in natural language, as well as wider use of NLP methods such as pre-trained language models and parsers to inform RL agents about the structure of the world. In the light of rapid progress in large language and vision-language models, as well as development of many new environments since the publication of the survey, we revisit our predictions and recommendations.

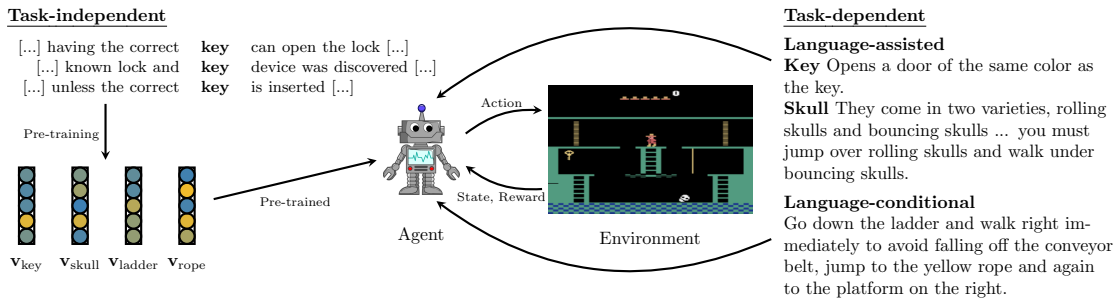


Figure 3.1: Illustration of different roles and types of natural language information in reinforcement learning. We differentiate between *language-conditional* setting in which language is a part of the task formulation (*e.g.* natural language instructions that specify the goal or reward), and *language-assisted* setting where language information is not necessary to solve the task but can assists learning (*e.g.* by providing information about the environment dynamics). The language information itself can be *task-dependent*, *i.e.* specific to the task such as tutorials or instructions, or *task-independent*, for instance, conveying general priors about the world through pre-trained language representations.

3.2 Use of Natural Language in RL

In reviewing efforts that integrate language in RL we highlight work that develops tools, approaches, or insights that we believe may be particularly valuable for improving the generalization or sample efficiency of learning agents through the use of natural language. As illustrated in Figure 3.1, we separate the literature into **language-conditional** RL (in which interaction with language is necessitated by the problem formulation itself) and **language-assisted** RL (in which language is used to facilitate learning). The two categories are not mutually exclusive, in that for some language-conditional RL tasks, NLP methods or additional textual corpora are used to assist learning (Bahdanau et al., 2019; Goyal et al., 2019b).

To easily acquire data and constrain the difficulty of problems considered, the majority of these works use synthetic language (automatically generated from a simple grammar and limited vocabulary) rather than language generated by humans. These often take the form of simple templates, *e.g.* "what colour is <object> in <room>", but can be extended to more complex templates with relations and multiple clauses (Chevalier-Boisvert et al., 2019).

3.2.1 Language-conditional RL

We first review literature for tasks in which integrating natural language is unavoidable, *i.e.*, when the task itself is to interpret and execute instructions given in natural language, or natural language is part of the state and action space. We argue in (§3.3.1) that approaches to such tasks can also be improved by developing methods that enable transfer from general and task-specific textual corpora. Methods developed for language-conditional tasks are relevant for language-assisted RL as they both deal with the problem of grounding natural language sentences in the context of RL. Moreover, in tasks such as following sequences of instructions, the full instructions are often not necessary to solve the underlying RL problem but they assist learning by structuring the policy (Andreas et al., 2017b) or by providing auxiliary rewards (Goyal et al., 2019b).

Instruction Following

Instruction following agents are presented with tasks defined by high-level (sequences of) instructions. We focus on instructions that are represented by (at least somewhat natural) language, and may take the form of formal specifications of appropriate actions, of goal states (or goals in general), or of desired policies. Effective instruction following agents execute the low-level actions corresponding to the optimal policy or reach the goal specified by their instructions, and can generalize to unseen instructions during testing. This might include understanding of spatial references or learning which sequence of actions corresponds to the verb contained in the instruction. Even for relatively simple environments with only a few types of objects and interactions available, learning to generalize over compositional instructions can be very data intensive (Chevalier-Boisvert et al., 2019).

In a typical instruction following problem, the agent is given a description of the goal state or of a preferred policy as a proxy for a description of the task (MacMahon et al., 2006; Kollar et al., 2010). Some work in this area focuses on simple object manipulation tasks (Wang et al., 2016; Bahdanau et al., 2019), while other work focuses on 2D or 3D navigation tasks where the goal is to reach a specific entity.

Entities might be described by predicates (“Go to the red hat") (Hermann et al., 2017; Chaplot et al., 2018) or in relation to other entities (“Reach the cell above the westernmost rock.") (Janner et al., 2018; Chen et al., 2018). Earlier approaches use object-level representation and relational modeling to exploit the structure of the instruction in relation to world entities, parsing the language instruction into a formal language (Kuhlmann et al., 2004; Chen & Mooney, 2011; Artzi & Zettlemoyer, 2013; Andreas & Klein, 2015). With the developments in deep learning, a common approach has been to embed both the instruction and observation to condition the policy directly (Mei et al., 2016; Hermann et al., 2017; Chaplot et al., 2018; Janner et al., 2018; Misra et al., 2017; Chen et al., 2018).

An agent can be tasked with a sequence of instructions (or primitives) and rewarded only once all of the instructions have been completed. The line of work involving sequences of instructions has strong ties to Hierarchical RL (Barto & Mahadevan, 2003), with individual sentences or clauses from instructions corresponding to sub-tasks (Branavan et al., 2010). For context on the scale of typical tasks in this category, in (Andreas et al., 2017b; Oh et al., 2017b) the length of sequences is up to 20 primitives, and the instructions are sampled from a small set of object-verb pairs. When the vocabulary of instructions is sufficiently simple, an explicit options policy can be constructed that associates each task description with its own modular sub-policy (Andreas et al., 2017b). A more flexible approach is to use a single policy that conditions on the currently executed instruction, allowing some generalization to unseen instructions (e.g. Mei et al., 2016; Graves et al., 2016; Wang et al., 2016; Hermann et al., 2017; Oh et al., 2017b); (Mei et al., 2016; Oh et al., 2017b). However, current approaches of this form require first pre-training the policy to interpret each of the primitives in a single-sentence instruction following setting. The internal compositional structure of instructions can then be exploited in various ways. For example, Oh et al. (2017b) achieve generalization to unseen instructions by forcing instruction embeddings to capture analogies, *e.g.*, [Visit,X] : [Visit,Y] :: [Pick up,X] : [Pick up,Y]. Later, a meta-controller is trained to execute on the entire

sequence, choosing at each timestep which instruction should be executed. The meta-controller also enabled the agent to handle simple interruptions.

At the time of publishing Luketina et al. (2019), the use of human-generated instructions, as opposed to synthetic language, was not a standard in the field (Hermann et al., 2017), although natural language instructions are used in MacMahon et al. (2006); Bisk et al. (2016); Misra et al. (2017); Janner et al. (2018); Goyal et al. (2019b); Wang et al. (2018).

Since then, works such as (Hill et al., 2020; Marzoev et al., 2020) demonstrated how through the use of pre-trained sentence embeddings, the agents trained to follow synthetically-generated templated instructions ("Put the blue box next to the door.") can generalize to natural instructions given by humans ("I want the dark blue crate adjacent to the opening."). In Marzoev et al. (2020), the human instruction is interpreted by first finding the closest synthetic instruction in the embedding space, which is then given to the agent. In Hill et al. (2020), synthetic instructions during training and human instructions during evaluation are both represented with the same frozen pre-trained sentence embedding.

Rewards from Instructions

Another use of instructions is to induce a reward function for RL agents or planners to optimize. This is relevant when the environment reward is not available to the agent at test time, but is either given during training (Tellex et al., 2011) or can be inferred from (parts of) expert trajectories. To apply instruction following more broadly, there needs to be a way to automatically evaluate whether the task specified by the instruction has been completed. The work addressing this setting is influenced by methods from the inverse reinforcement learning (IRL) literature (Ziebart et al., 2008; Ho & Ermon, 2016). A common architecture consists of a reward-learning module that learns to ground an instruction to a (sub-)goal state or trajectory segment and is used to generate a reward for a policy-learning module or planner.

When full demonstrations are available, the reward function can be learned using standard IRL methods like maximum entropy IRL (Ziebart et al., 2008)

as in Fu et al. (2019), or maximum likelihood IRL (Babes et al., 2011) as in MacGlashan et al. (2015), who also learn a joint generative model of rewards, behaviour, and language. Otherwise, given a dataset of goal-instruction pairs, as in Bahdanau et al. (2019), the reward function is learned through an adversarial process similar to GAIL Ho & Ermon (2016). For a given instruction, the reward-learning module aims to discriminate goal states from the states visited by the policy (assumed non-goal), while the agent is rewarded for visiting states the discriminator cannot distinguish from the goal states. Williams et al. (2018) learn a parser to transform instructions to a logical form which can be used to deduce the goal state, while MacGlashan et al. (2015).

A related line of work involves using language annotations and instructions to provide auxiliary rewards (see Section 3.2.2). Because language is not the part of the problem formulation in those works, we categorize it under language-assisted RL.

Language in the Observation and Action Space.

Environments that use natural language as a first-class citizen for driving the interaction with the agent present a strong challenge for RL algorithms. Using natural language requires common sense, world knowledge, and context to resolve ambiguity and cheaply encode information Mey (1993). Furthermore, linguistic observation and action spaces grow combinatorially as the size of the vocabulary and the complexity of the grammar increase. For instance, compare the space of possible instructions when following cardinal directions (*e.g.* "go north") with reaching a position that is described in relative terms (*e.g.* "go to the blue ball south west of the green box"). Furthermore, when tasks require multiple sentences or paragraphs to facilitate meaningful interactions, this compositionality introduces additional complexity.

Text Games. Text games, such as Zork (Infocom, 1980), are easily framed as RL environments and make a good testbed for structure learning, knowledge extraction, and transfer across tasks (Branavan et al., 2012). Both state and action spaces of text games are in natural (or synthetic) language, as in the example sequence from

the game Zork Infocom (1980): s_t = "You are standing in an open field west of a white house. There is a small mailbox here.", a_t = "open mailbox", s_{t+1} = "Opening the small mailbox reveals a leaflet", a_{t+1} = "take leaflet". (DePristo & Zubek, 2001; Narasimhan et al., 2015; Yuan et al., 2018) observe that when the action space of the text game is constrained to verb-object pairs, decomposing the Q-function into separate parts for verb and object provides enough structure to make learning more tractable. However, they do not show how to scale this approach to action-sentences of arbitrary length. To facilitate the development of a consistent set of benchmarks in this problem space, Côté et al. (2018) propose *TextWorld*, a framework that allows the generation of instances of text games that behave as RL environments. They note that existing work on word-level embedding models for text games (e.g. (Kostka et al., 2017; Fulda et al., 2017)) achieve good performance only on easy tasks. Pre-trained language models are used in works like (Yao et al., 2020; Singh et al., 2022), to either provide action candidates which are then re-ranked by the RL agent (Yao et al., 2020) or as intialization of the state representation (Singh et al., 2022).

Embodied Question Answering. Other examples of settings where agents are required to interact using language include *dialogue systems* and *question answering* (Q&A). The two have been a historical focus in NLP research and are extensively reviewed by Chen et al. (2017b) and Bouziane et al. (2015) respectively. Recent work on visual Q&A (VQA) have produced a first exploration of multi-modal settings in which agents are tasked with performing both visual and language-based reasoning (Antol et al., 2015; Johnson et al., 2017; Massiceti et al., 2018). Embodied Q&A (EQA) extends this setting, by requiring agent to explore and navigate the environment in order to answer queries (Das et al., 2018a; Gordon et al., 2018), for example, "How many mugs are in the kitchen?" or "Is there a tomato in the fridge?". By employing rich 3D environments, EQA tasks require agents to carry out multi-step planning and reasoning under partial observability. However, since in the existing work the agents only choose from a limited set of short answers

instead of generating an arbitrary length response, so far such tasks have been very close to the instruction following setting.

Autonomous Agent Systems. One of the most significant developments since the publication of the survey (Luketina et al., 2019) are autonomous agent systems (Gravitas, 2023; Yao et al., 2023; Boiko et al., 2023): AI systems consisting of a LLM and scaffolding (software wrapper handing formatting of inputs and outputs of LLM and execute commands in external APIs). The scaffold enables the LLM to operate autonomously, interact with the environment (e.g., databases, bash terminals, documents), and execute long multi-step plans. Following an initial prompt or instruction, each of the language model's responses constitutes an action. The actions are interpreted by the scaffolding, which then enacts the action and converts the result into text, which can then be appended to the model's context window. This, in turn, leads to the generation of the next action. The ability to use external programs through scaffolding is also referred to as tool use. While both autonomous agent systems and agents in the RL literature involve entities interacting autonomously with the environment, there are notable differences between the two terms. Unlike RL agents, autonomous agents typically do not receive rewards or learn from experience (although they can, as in (Parisi et al., 2022; Schick et al., 2024)), instead leveraging the knowledge and reasoning capabilities of language models to operate independently, alongside instruction-tuning which enables LLMs to follow user instructions (nowadays a standard feature of state-of-the-art LLMs (Ouyang et al., 2022; Bai et al., 2022)). Given the right scaffolding, autonomous agent systems can interact with any domain in which the state can be expressed as language or code (as well as images, if multi-modal LLMs such as OpenAI (2023b) and Gemini Team et al. (2023) are used). To study and compare the behavior of autonomous agent systems, a large number of environments that require reasoning, tool-use and multi-step interaction have been developed (Liu et al., 2024). These environments include tasks like simulated web-interaction (Yao et al., 2022), multi-step question answering (Mialon et al., 2023b) and debugging codebases (Jimenez

et al., 2023). For a more comprehensive overview of work on autonomous agent systems, readers can refer to the review by (Mialon et al., 2023a).

3.2.2 Language-assisted RL

In this section, we consider work that explores how knowledge about the structure of the world can be transferred from natural language corpora and methods into RL tasks, in cases where language understanding is not needed to solve the task. Textual information in itself or through LLM can assist learning by providing a source of auxiliary rewards, annotating states or entities in the environment or structuring the policies.

Language for Auxiliary Rewards

In this section, we cover several works exploring the use of language annotations or representations for generation of auxiliary rewards in sparse reward environments. Goyal et al. (2019b) and Wang et al. (2018) use auxiliary reward-learning modules trained offline to predict whether trajectory segments correspond to natural language annotations of expert trajectories. Agarwal et al. (2019) perform a meta-optimization to learn auxiliary rewards conditioned on features extracted from instructions. The auxiliary rewards are meta-learned with an objective of increasing performance on the true objective after being used for a policy update.

More recent works such as Mu et al. (2022) and Tam et al. (2022) use pre-trained language representations of annotations or annotations generated by pre-trained vision-language models as state representations in state-based intrinsic exploration methods such as Burda et al. (2018) and Campero et al. (2020). The key idea in these works is that language captures relevant abstractions in the environment, hence providing a more meaningful measure of novelty compared to raw state representations (this problem is also a focus of Chapter 5). This key idea is also a motivation in Du et al. (2023), where given a textual description of the current state but without knowledge of the task, LLM proposes a series of potentially useful

goals. The agent is then rewarded for generating transitions whose description matches the proposed goals.

In Klissarov et al. (2023), an intrinsic reward model is obtained by prompting an LLM to express preferences over the pairs of captions from a dataset of environment observations. In Nethack Küttler et al. (2020), a challenging sparse reward game that provides captions in certain states, the inclusion of this intrinsic reward leads to significant improvements over training with only extrinsic reward.

Language for Communicating Domain Knowledge

In settings more general than instruction following, any kind of textual corpora containing task-specific information could be available. Such text may contain advice regarding the policy an agent should follow or information about the environment dynamics (see Figure 3.1). Unstructured and descriptive (in contrast to instructive) textual information is generally more abundant and can be found in wikis, manuals, books, or on the internet. However, using such information requires (i) retrieving useful information for a given context and (ii) grounding that information to observations.

Eisenstein et al. (2009) learn abstractions in the form of conjunctions of predicate-argument structures that can reconstruct sentences and syntax in task-relevant documents using a generative language model. These abstractions are used to obtain a feature space that improves imitation learning outcomes. Labutov et al. (2019) describe a procedure for interactively improving a semantic parser with user feedback provided in the form of natural language.

Branavan et al. (2012) learn a Q-function that improves Monte-Carlo tree search planning for the first few moves in Civilization II, a turn-based strategy game, while accessing the game's natural language manual. Features for their Q-function depend on the game manual via learned sentence-relevance and predicate-labeling models whose parameters are optimised only by minimizing the Q-function estimation error. Due to the structure of their hand-crafted features (some of which match states and actions to words, *e.g.* `action == irrigate AND action-word == "irrigate"`),

these language processing models nonetheless somewhat learn to extract relevant sentences and classify their words as relating to the state, action, or neither. More recently, Wu et al. (2023) investigate the use of game instruction manuals, in Atari environment (Mnih et al., 2013). Utilizing several major developments in NLP since Branavan et al. (2012), the authors proposes a framework consisting a reading module (a smaller LLM fine-tuned for extractive QA) to extract and summarize relevant information from the manual, and a reasoning module (a larger QA model prompted with a templated question) to provide the auxiliary reward for the agent. Ground-truth state variables are used to provide textual state descriptions as context for the two QA modules. The authors also examine a more general setting, by creating state descriptions through a combination of off-the-shelf open-source object detection and image annotation models, but find this approach unreliable due to poor generalization of the off-the-shelf methods on the Atari domain.

Narasimhan et al. (2018) investigate planning in a 2D game environment where properties of entities in the environment are annotated by natural language (*e.g.* the 'spider' and 'scorpion' entities might be annotated with the descriptions "randomly moving enemy" and "an enemy who chases you", respectively). Descriptive annotations facilitate transfer by learning a mapping between the annotations and the transition dynamics of the environment. These annotations are embedded in a grid corresponding to each cell of the 2D environment alongside an entity identity feature, simplifying the task of grounding. Hanjie et al. (2021) extend this line of work by removing grounding assistance, instead learning the mapping between observed entities and their references in text purely through interaction with the environment. Their approach relies on being able to procedurally generate a large number of toy environments, while collecting a dataset of 5,000 natural language entity descriptions.

Language for Structuring Policies

One use of natural language is communicating information about the state and/or dynamics of an environment. As such it is an interesting candidate for construct-

ing priors on the model structure or representations of an agent. This could include shaping representations towards more generalizable abstractions, making the representation space more interpretable to humans, or efficiently structuring the computations within a model.

Andreas et al. (2016) propose a neural architecture that is dynamically composed of a collection of jointly-trained neural modules, based on the parse tree of a natural language prompt. While originally developed for visual question answering, Das et al. (2018b) and Bahdanau et al. (2019) successfully apply variants of this idea to RL tasks. Andreas et al. (2018) explore the idea of natural language descriptions as a policy parametrization in a 2D navigation task adapted from Janner et al. (2018). In the pre-training phase, the agent learns to imitate expert trajectories conditioned on instructions. By searching in the space of natural language instructions, the agent is then adapted to the new task where instructions and expert trajectories are not available.

The hierarchical structure of natural language and its compositionality make it a particularly good candidate for representing policies in hierarchical RL. Andreas et al. (2017b) and Shu et al. (2018) can be viewed as using language (rather than logical or learned representations) as policy specifications for a hierarchical agent. Hu et al. (2019) consider generated natural language as a representation for macro actions in a real-time strategy game environment based on Tian et al. (2017). In an IL setting, a meta-controller is trained to generate a sequence of natural language instructions. Simultaneously, a base-controller policy is trained to execute these generated instructions through a sequence of actions.

Language Models as Planners In a large number of recent works (Huang et al., 2022b; Ahn et al., 2022; Song et al., 2023; Huang et al., 2023; Nottingham et al., 2023), the semantic knowledge of the world and reasoning abilities of LLMs are leveraged for planning in embodied instruction following tasks. Upon receiving an instruction (e.g., “I spilled my drink, can you help me?"), the language model comes up with a high-level plan that consists of low-level instructions (e.g., “I will: (1) find

a sponge, (2) pick up the sponge, (3) come to you."), each of which can be executed by the lower-level instruction following policy. Language models in these works are used in a zero-shot manner, without fine-tuning on the sequential decision-making tasks or being used during training of the low-level policies. Some of the challenges of using LLM planners include ensuring generation of plans that the lower-level policy can recognize and execute, and dynamically adapting the plan based on other kinds of information (e.g., visual observations or user feedback). Ahn et al. (2022) selects each lower-level instruction by combining its probability of usefulness (indicated by the probability of instruction under the LLM), with the probability of its successful execution (indicated by its value function). Huang et al. (2023) find that providing the LLM planner with several other sources of textual feedback (including success detection, scene description and human feedback) improves the performance over Ahn et al. (2022).

Language Models in the Architecture In contrast to works on LLMs as planners, the following works incorporate language models and representations directly into the learned policy. Li et al. (2022) learn a policy that consists of a sequence tokenizer, a pre-trained language model (transformer) and a task-specific action prediction network. They find that the use of language pre-trained (in contrast to using a newly initialized) transformer improves generalization on instruction following tasks like BabyAI (Chevalier-Boisvert et al., 2019) and VirtualHome (Puig et al., 2018). In Paischer et al. (2022), a frozen pre-trained language model is used to encode the history of observations in partially observable MDPs, resulting in at the time state-of-the-art results on Minigrid (Chevalier-Boisvert et al., 2018) and Procgen (Cobbe et al., 2019) environments.

3.3 Trends for Natural Language in RL

The preceding sections surveyed the literature exploring how natural language can be integrated with RL. Before 2019, several trends were evident: (i) studies for language-conditional RL were more numerous than for language-assisted RL,

(ii) learning from task-dependent text was more common than learning from task-independent text, (iii) within work studying transfer from task-dependent text, only a handful of papers study how to use unstructured and descriptive text, (iv) there were only a few papers exploring methods for structuring internal plans and building compositional representations using the structure of language, and finally (v) natural language, as opposed to synthetically generated languages, was not the standard in research on instruction following.

To advance the field, we argued that more research effort should be spent on learning from naturally occurring text corpora in contrast to instruction following. While learning from unstructured and descriptive text is particularly difficult, it has a much greater application range and potential for impact. Moreover, we argued for the development of more diverse environments with real-world semantics. The tasks used before 2019, relied on small and synthetic language corpora and were too artificial to significantly benefit from transfer from real-world textual corpora. In addition, we emphasized the importance of developing standardized environments and evaluations for comparing and measuring the progress of models that integrate natural language into RL agents.

We believed that several factors make focusing such efforts worthwhile: (i) progress in pre-training language models at the time, (ii) general advances in representation learning, as well as (iii) development of tools that make constructing rich and challenging RL environments easier.

Some significant work, especially in language-assisted RL, has been done prior to the surge of deep learning methods (Eisenstein et al., 2009; Branavan et al., 2012), and is worth revisiting. In addition, we encouraged the reuse of software infrastructure, e.g. (Bahdanau et al., 2019; Côté et al., 2018; Chevalier-Boisvert et al., 2019) for constructing environments and standardized tests.

3.3.1 Learning from Text Corpora in the Wild

The web contains abundant textual resources that provide instructions and how-to's.[2] For many games (which are often used as testbeds for RL), detailed walkthroughs and strategy guides exist. We believe that transfer from task-independent corpora could also enable agents to better utilize such task-dependent corpora. Preliminary results that demonstrate zero-shot capabilities (Radford et al., 2019) suggest that a relatively small dataset of instructions or descriptions could suffice to ground and consequently utilize task-dependent information for better sample efficiency and generalization of RL agents.

Task-independent Corpora

Natural language reflects human knowledge about the world (Zellers et al., 2018). For instance, an effective language model should assign a higher probability to "get the green apple from the tree behind the house" than to "get the green tree from the apple behind the house". Harnessing such implicit commonsense knowledge captured by statistical language models could enable transfer of knowledge to RL agents.

For the short-term, in Luketina et al. (2019) we anticipated more use of pre-trained word and sentence representations for research on language-conditional RL. For example, consider instruction following with natural language annotations. Without transfer from a language model (or another language grounding task as in Yu et al. (2018)), instruction-following systems cannot generalize to instructions outside of the training distribution containing unseen synonyms or paraphrases (*e.g.* "fetch a stick", "return with a stick", "grab a stick and come back"). While pre-trained word and sentence representations alone will not solve the problem of grounding an unseen object or action, they do help with generalization to instructions with similar meaning but unseen words and phrases. In addition, we believed that learning representations for transferring knowledge about analogies, going beyond using analogies as auxiliary tasks (Oh et al., 2017b) will play an important role in generalizing to unseen instructions.

[2] *e.g.* `https://www.wikihow.com/` or `https://stackexchange.com/`

As pre-trained language models and automated question answering systems become more capable, we advocated as a promising research direction, studying agents that can query knowledge more explicitly. For example, during the process of planning in natural language, an agent that has a pre-trained language model as sub-component could let the latter complete “to open the door, I need to..." with “turn the handle". Such an approach could be expected to learn more rapidly than *tabula rasa* reinforcement learning. However, such agents would need to be capable of reasoning and planning in natural language, which is a related line of work (see Language for Structuring Policies in §3.2.2).

Task-dependent Corpora

Research on transfer from descriptive task-dependent corpora is promising due to its wide application potential. In (Luketina et al., 2019), we argued that it also requires development of new environments, as early research may require access to relatively structured and partially grounded forms of descriptive language similarly to Narasimhan et al. (2018). One avenue for early research, we argued, is developing environments with relatively complex but still synthetic languages, providing information about environmental dynamics or advice about good strategies. For example, in works studying transfer from descriptive task-dependent language corpora (Narasimhan et al., 2018), natural language sentences could be embedded using representations from pre-trained language models. We argued that integrating and fine-tuning pre-trained information retrieval and machine reading systems similar to (Chen et al., 2017a) with RL agents that query them, could help in extracting and utilizing relevant information from unstructured task-specific language corpora, such as game manuals as used in Branavan et al. (2012). Lastly, we believed pre-trained language models could be used in RL agents to express their planning in natural language and, specifically, to regularize their plans to action sequences that would likely be encountered in the real world.

Revisiting Recommendations

Since the release of the survey paper, the use of LLMs for enabling reasoning and planning in embodied control has been explored by a large number of works (see Section 3.2.2). These works demonstrate the successful transfer of commonsense understanding from the LLMs to practical robotics problems. As anticipated, we have also seen the use of language embeddings for generalization to unseen synonyms (specifically, from synthetic to human language) (Hill et al., 2020; Marzoev et al., 2020).

The grounding remains a significant obstacle to using information from task-dependent corpora (Hanjie et al., 2021; Wu et al., 2023). In terms of specialized Q&A systems, the most promising approach currently is retrieval-augmented systems (Lewis et al., 2020). At the same time, the LLMs are trained with increasingly large context windows, with some of the latest LLM models like Gemini (Gemini Team et al., 2023) supporting up to 1 million token context windows. For some applications, this means just adding the relevant documents in the context window may be sufficient. Incorporating querying of state-of-the-art language models, if used during training of RL agents at their current sample efficiency, is limited by high inference costs, though this can be circumvented in some cases by using offline data to create a smaller reward model as in Klissarov et al. (2023). On the other hand, smaller language models at the time of writing, lack the knowledge and reasoning abilities of large models. Partly for this reason, current applications of state-of-the-art LLMs in sequential decision-making are mostly constrained to zero-shot settings (see Section 3.2.2).

While still relevant for enabling sim-to-real and learning control, the development of large pre-trained vision-language models (VLM) (Radford et al., 2021; Alayrac et al., 2022; Chen et al., 2023) and multi-modal language models such as Gemini (Gemini Team et al., 2023) and GPT-4v (OpenAI, 2023b), on visual domains provide a useful supplement to learning language grounding purely through interaction. In (Stone et al., 2023), a frozen VLM is used to identify the location of the relevant

object given templated instruction, which is then incorporated in the instruction-following policy. The authors find that resulting model demonstrates zero-shot generalization to novel objects and environments. In (Zitkovich et al., 2023), VLM has been fine-tuned on instruction following robotic trajectories (in addition to original web data). The use of VLM resulted in significantly improved generalization, including the ability to follow instructions not seen in the robotic data (e.g., fetching an object not seen in robotic data), as well as simple reasoning in response to user commands (e.g., picking up the smallest object).

We expect that VLMs and multi-modal language models will play an increasingly large role in enabling language grounding, alongside LLMs supporting sequential decision-making models with reasoning and planning abilities. At the same time, autonomous agent systems (as described in Section 3.2.1) will improve capabilities through the development of better scaffolds and fine-tuning of underlying LLMs for tool use.

3.3.2 Towards Diverse Environments with Real-World Semantics

One of the central promises of language in RL is the ability to rapidly specify and help agents adapt to new goals, reward functions, and environment dynamics. This capability is not exercised at all by standard RL benchmarks like strategy games (which typically evaluate agents against a single or small number of fixed reward functions). It is evaluated in only a limited way by existing instruction following benchmarks, which operate in closed-task domains (navigation, object manipulation, etc.) and closed worlds. The simplicity of these tasks is often reflected in the simplicity of the language that describes them, with small vocabulary sizes and multiple pieces of independent evidence for the grounding of each word.

Real natural language has important statistical properties, such as the power-law distribution of word frequencies (Zipf, 1949) which does not appear in environments with synthetically generated language and small numbers of entities. Without environments that encourage humans to process (and force agents to learn from)

complex composition and the "long tail" of lexicon entries, we cannot expect to hope that RL agents will generalize at all outside of *closed-world tasks*.

In Luketina et al. (2019), we suggested extensions of the 3D house simulation environments as a starting point (Gordon et al., 2018; Das et al., 2018a; Yan et al., 2018; Savva et al., 2019) some of which already support generation and evaluation of templated instructions. Moreover, these environments contain more real-world semantics (e.g. a microwave can be opened, a car is in the garage). Since 2019, such environments have become richer and more realistic (Shridhar et al., 2020; Puig et al., 2024; Lynch et al., 2023). Some of these environments have since been used as test-beds for benchmarking methods using LLMs for planning (Huang et al., 2022b; Song et al., 2023; Wong et al., 2023).

Another option suggested in Luketina et al. (2019) is open-world video games like Minecraft (Johnson et al., 2016), in which users are free to assemble complex structures from simple parts, and thus have an essentially unlimited universe of possible objects to describe and goals involving those objects. MineDojo (Fan et al., 2022) is a recent project, providing an open-source RL benchmark with thousands of open-ended tasks and a database of videos, tutorials as well as online discussions and wiki pages. The authors also fine-tune a vision-language model on Minecraft data (MineCLIP), and use it as a reward model by rewarding behaviors that align with language instruction. The reward model based on MineCLIP agrees with ground-truth human judgement, even out-performing the hand-engineered reward functions. Versions of Minecraft have been also used as a test-bed for studying planning with LLMs in (Nottingham et al., 2023; Wong et al., 2023; Wang et al., 2023).

In Luketina et al. (2019), we anticipated that, as core machine learning tools for learning from feedback and demonstrations become sample-efficient enough to use in the real world, the approaches combining language and RL will find applications as wide-ranging as autonomous vehicles, virtual assistants and household robots. Over the past years, we have started to see these predictions come to fruition.

3.4 Conclusion

The currently predominant way RL agents are trained restricts their use to environments where all information about the policy can be gathered from directly acting in and receiving reward from the environment. This *tabula rasa* learning results in low sample efficiency and poor performance when transferring to other environments. Utilizing natural language in RL agents could drastically change this by transferring knowledge from natural language corpora to RL tasks, as well as between tasks, consequently unlocking RL for more diverse and real-world tasks. While there is a growing body of papers that incorporate language into RL, most of the research effort has been focused on simple RL tasks and synthetic languages, with highly structured and instructive text.

To realize the potential of language in RL, in (Luketina et al., 2019) we advocated for more research into learning from unstructured or descriptive language corpora, with a greater use of NLP tools like pre-trained language models. The use of pre-trained language models in particular, turned out to be a very successful approach over the past couple of years. We also advocated for the development of more challenging environments that reflect the semantics and diversity of the real world, which while still relevant, diminished in importance due to the rise of vision-language and multi-modal language models.

4

Compositional Interfaces for Compositional Generalization

4.1 Introduction

In recent years, there has been remarkable successes with scaling model and data sizes. Across a wide variety of domains, the state-of-the-art is dominated by large models (pre-)trained on billions of samples (Brown et al., 2020; Goyal et al., 2021; Bommasani et al., 2021). This also holds true in the setting of multi-domain "generalist" agents, that can integrate perceptual information across multiple modalities, and can accomplish a variety of tasks (Reed et al., 2022; Shridhar et al., 2023).

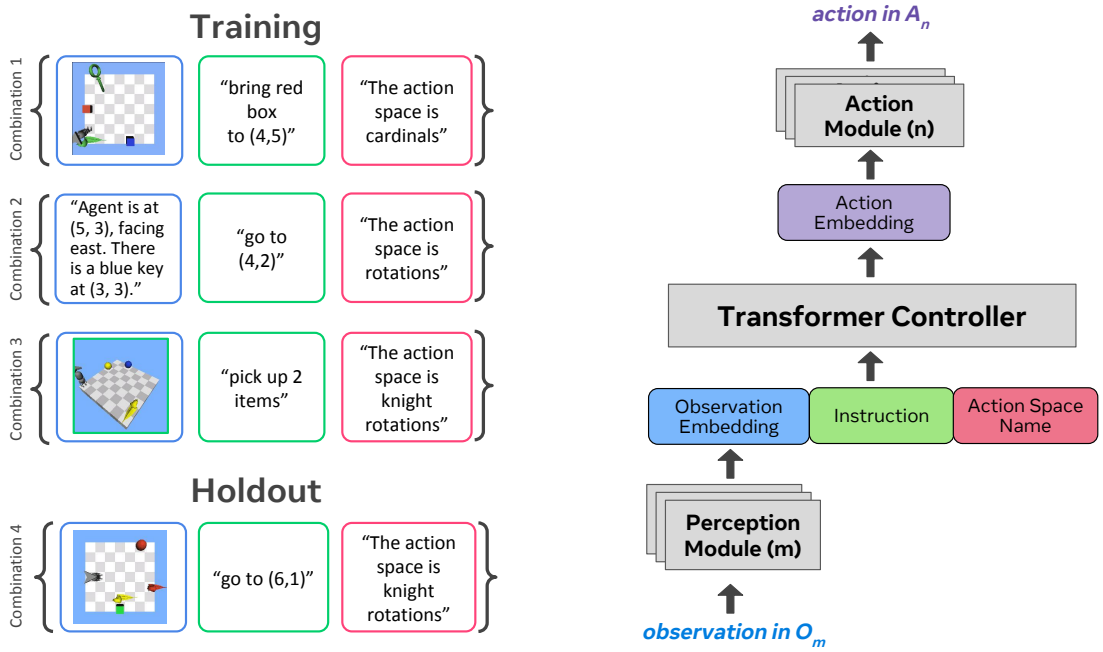
In this multi-domain setting, transferring common knowledge between domains while respecting the particularities of each domain is still not a solved problem. For example, in the setting of agents that are either virtually or physically embodied, one of the most important special cases is simulation-to-real transfer. Practitioners would like to use gradient-based end-to-end learned controllers, but it is difficult to collect large amounts of training data on a physical robot in the real world. While there has been great progress in some tasks (Wijmans et al., 2019), driven in large part by ever higher-fidelity simulations (Savva et al., 2019; Shen et al.,

2021), there are still no out-of-the-box solutions for generic sim-to-real transfer[1]. More generally, one would like to be able to train agents as much as possible in domains where training is cheap, and deploy after minimal training in domains where training is expensive. Even more generally, between domain transfer may allow sample efficiency via composition and abstraction. For example, a wheeled robot, a legged robot, and a digital assistant "embodied" in AR glasses all might be able to share some knowledge about indoor navigation despite differences in their locomotion. However, it is not possible to entirely abstract away all these differences- the wheeled robot cannot traverse the same terrain as the human with AR glasses.

Based on the above-cited results in large-scale sequence modeling, one might wonder if researchers need to worry explicitly about transfer; maybe it is better to just scale token-based monolithic models like (Reed et al., 2022), or scale language models and some task/domain-specific models, with inter-model text interfaces as in (Ahn et al., 2022; Zeng et al., 2022). We are sympathetic to these viewpoints, but as much as the bitter lesson has been that scale can be more important than good inductive bias, it has *also* been that optimizing end-to-end leads to the best results. Properly designed modular architectures can be both scalable, and allow end-to-end training (Pfeiffer et al., 2023). Furthermore, one of the benefits of modern attention-based architectures is that they are conducive to modular inductive biases without radical changes (Alayrac et al., 2022; Jaegle et al., 2021; Shridhar et al., 2023), and can differentiably interface various domains and still directly take advantage of pre-trained language models. Thus, we might hope to both encourage transfer through abstraction and composition, and allow end-to-end fine-tuning to handle the necessary details that cannot be abstracted; without giving up any of the benefits of large-scale pre-training.

The idea of composing the architecture from several specialized and some shared modules is not novel (Ngiam et al., 2011; Devin et al., 2017; Kaiser et al., 2017; Pfeiffer et al., 2022, 2023) and can be considered a fairly straightforward approach to the problem of multi-modal multi-task learning. However, whether such an

[1] see on Robot Learning (2022) for lively debates on this issue.



(a) Dataset Formulation

(b) COIN Architecture

Figure 4.1: Illustration of the dataset formulation and the COIN (**Co**mpositional **In**terfaces) architecture for compositional generalization. **(a)** Each environment instance is defined by a tuple (O_m, A_n, I_k): a combination of observation O_m, action A_n and instruction I_k spaces. The agent is trained on data from a subset of all possible combinations, with the expectation of generalizing to combinations not included in the training dataset. **(b)** The agent architecture consists of: perception modules (one for each observation space), action modules (one for each action space) and a controller (shared across all environment instances). The controller takes the observation embedding, instruction and action space identifier as input, while outputting action embedding. When acting in an environment consisting of observation space O_m and action space A_n, m-th perception module is used to create the observation embedding and n-th action module is used to predict action from the action embedding.

approach can learn mappings to a meaningful and useful shared space is not given, and whether it can lead to generalization to unseen combinations of modalities and tasks has not been studied.

In Luketina et al. (2023), we provide one such study, focusing on measuring the effectiveness of modular architectures for compositional generalization and transfer learning in embodied instruction-following setting. We develop an environment that allows us to independently vary perceptual modalities, as well as action and instruction specifications, and use it to carefully analyze the agent's performance in these compositions. Our experiments demonstrate that modular architectures

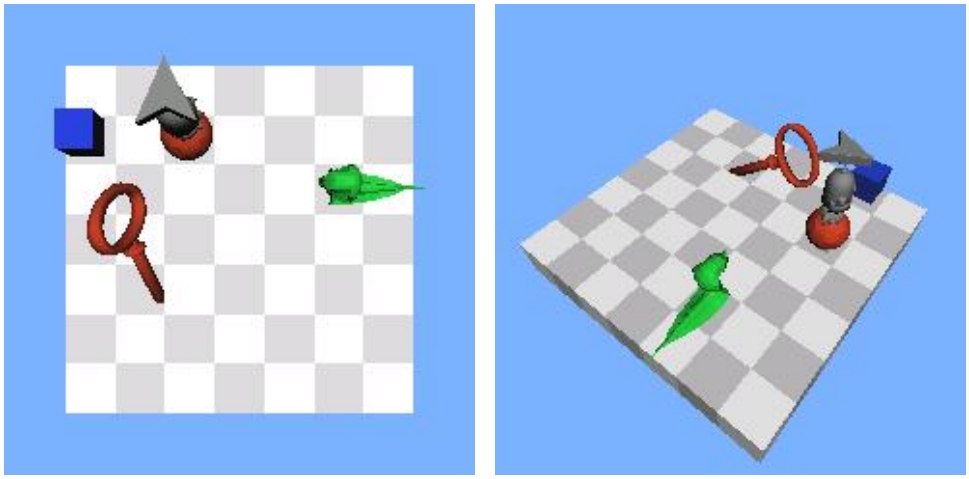
enable zero-shot transfer to held-out combinations of perception/action/instruction-spaces, and fast adaptation to new perceptual spaces with or without freezing of the weights shared across tasks.

4.2 Setting

Our goal is to solve tasks defined by an environment instance (O_m, A_n, I_k), which is constructed by combining m-th observation (O), n-th action (A) and k-th instruction (I) space.[2] Given an observation $o^{(m)}$ from space O_m, action space id n for A_n, and instruction $i^{(k)}$ from space I_k, the goal is to find a policy that will predict an optimal action $\pi(o^{(m)}, n, i^{(k)}) \rightarrow a \in A_n$. The agent is trained using imitation learning (Schaal, 1999) on a dataset of expert trajectories $\{D_{m,n,k}\}$ collected on (O_m, A_n, I_k). We make sure that during training, the agent will be trained on samples from environment instances containing at least one of each of the individual spaces O_m, A_n, and I_k, but not *all possible combinations* (O_m, A_n, I_k). This allows us to test compositional generalization by deploying the agent in environments containing unseen combinations, as demonstrated in Figure 4.1 (a). Alternatively, we can measure how quickly it adapts to newly added space.

Note that in this setting, generalization can mean two different things: (1) in-domain generalization where the agent is trained on trajectories from (O_m, A_n, I_k), but a particular test sample $(o^{(m)}, n, i^{(k)})$ is never seen due to random procedural generation of environments. And (2), compositional generalization where test samples are from environment combinations that never seen during training. For example, learning to predict the right action in action space $A_{n'}$ given observation $o^{(m')}$ and instruction $i^{(k')}$ when the training dataset does not contain samples from environment $(O_{m'}, A_{n'}, I_{k'})$. In this work, we are particularly interested in the compositional generalization to unseen combinations of spaces.

[2]The instruction space here denotes a set of instructions that share relevant structure, for example, instructions of the form "Go to <location>" and "Pick up <number> items" will form two different instruction spaces. Without the loss of generality, instead of sets of instructions, this setting can be also defined by sets of goals or reward functions that share structure.



(a) Top View Observation **(b)** Side View Observation

Figure 4.2: Illustration of the environment as represented by observation in **(a)** *Top View* and **(b)** *Side View* space. The agent (gray) is tasked with completing the instruction by navigating and interacting with objects in the grid. Each object is defined by a shape (ball, box, key, snake) and color (red, blue, green, yellow).

4.3 Environment

To study compositional generalization to unseen combinations of spaces, we construct a grid-world environment that supports multiple interfaces for observation and actions. The state is a 7x7 grid containing up to four different objects in addition to the agent itself. The objects can be picked up by the agent given that they are next to each other. The agent's inventory show object that are picked up, which later can be dropped down. Each object has a shape (box, ball, snake, key) and a color (red, green, yellow, blue). Each environment combination is constructed by selection on observation, action, and instruction space from one of the available options.

Observation spaces (O_m). There are six possible observation spaces, in which positions, shapes, and colors of the objects and agent in the grid are represented by: Text, Symbols, List, Grid, Top View, or Side View. These spaces are detailed in Table 4.1. Text space describes everything in human understandable language. Symbol space is similar, but uses compact symbols instead of words. To build an observation in List space, we represent everything with one-hot representation first. Then, for each object, we concatenate all its properties into a single vector.

Observation Space	Description
Text	A natural language description, e.g., "The agent is at (3, 5), facing east. There is a yellow snake at (2, 0). The agent has following items in the inventory: a blue box."
Symbol	A sequence of symbols, e.g., "A @ E x3y5. y S @ x2y0. I: b B."
List	A 2D tensor `o`, where `o[i] = concat([object_i_position, one_hot(object_i_shape), one_hot(object_i_color)])`
Grid	A 3D tensor `o` where the object at position (x, y) is indicated by `o[x, y] = concat([one_hot(object_shape) one_hot(object_color)])`
Top View	An image made by projecting 3D space from the top (see Figure 4.2 right).
Side View	An image made by projecting 3D space from the side (see Figure 4.2 left).

Table 4.1: Observation Spaces

Finally, we stack all such vectors from all object and the agent together to give a complete description of the state. Grid space also builds a vector for each object first, but then arranges them by their location instead, producing a 3D tensor. The remaining Top and Side view spaces are simply image rendering of the environment. These image spaces and Grid space assumes spatial location, which does not apply to inventory objects. As a workaround, we use List representation of inventory for those spaces. We made sure each observation contains sufficient information for completing the instruction, hence the tasks are fully observable.

Action spaces (A_n). Completing each of the instructions can be accomplished by using one of the five possible action spaces. The type of movement available varies between spaces, as described in Table 4.2. Additionally, each action space has three shared actions for picking and dropping objects, and indicating the episode is done (Pick, Drop, and Done actions respectively). Successful completion requires the agent to complete the instruction and then output Done action (selecting the Done action has no effect outside the goal state).

Action Space	Description
Cardinals	Move one step in one of the 4 cardinal directions (north, east, south, west)
Move NW	Move one step north, move one step west, or teleport to the south-east corner of the grid
Rotations	Rotate left, rotate right, or move one step forward in the direction of facing
Teleport Direction	Rotate left, rotate right, or teleport to a certain distance from the wall currently facing (0-6 steps from the wall)
Knight Rotations	Rotate left, rotate right, knight move left, or knight move right (i.e. two steps forward in the direction of facing + one step left or right).

Table 4.2: Action Spaces

Instruction spaces (I_k). In a given environment instance, the agent is tasked with completing an instruction from one of the eight possible instruction spaces. The simplest instruction, "Go to (x,y)", requires the agent to reach the specified location, while more complex instructions like "Pick up in order: red box, yellow snake, and green box" involves multiple steps and require the agent to distinguish shapes and colors. The full list of instruction spaces is given in Table 4.3. All instruction spaces involve manipulating objects and positions of the agent, with individual instructions being randomly sampled from the instruction space, while satisfying the constraint of instruction completion being possible given the initial state.

4.4 Architecture with Compositional Interfaces

The most straightforward modular architecture for this setting consists of three main components (demonstrated in Figure 4.1 (b)):

- the perception modules: mapping the observations of different modalities to a shared space,
- the controller: sharing knowledge across different combinations and incorporating instructions,

Instruction Space	Description
Go To	The agent needs to move to a randomly sampled location. E.g., "Go to (1, 3)."
Pickup N	The agent needs to pick up exactly N objects, where N is a number randomly sampled between 1 and # objects in the environment. E.g., "Pickup 2 objects."
Pickup Color	Pick up one randomly chosen object specified by color. E.g., "Pick up one red item."
Bring Shape	Bring one randomly chosen object specified by shape to a randomly chosen target location. E.g., "Bring one key to (4, 4)."
Bring Conditional	Bring one item to one of the 4 corners, depending on the shape or color of the object. Whether the target corner is determined by shape or color is randomly sampled. E.g., "Bring one item to shape location."
Bring Object	Bring an item specified by a randomly chosen shape and color to a randomly chosen target location. E.g., "Bring green snake to (6, 2)."
Pickup in Order	Pick up items in a specified random order determined by the color and shape of the objects. E.g., "Pick up in order: red box, yellow snake, and green box."
Sort by Property	Move all items of the same color or shape within 1 step of each other. Whether the items should be moved based on shape or color is randomly sampled. E.g., "Sort items by color."

Table 4.3: Instruction Spaces

- the action modules: mapping the shared action embedding into specific actions.

We refer to it as COIN (**Co**mpositional **In**terfaces) architecture throughout the paper.

There is a different perception module for each observation space and a different action module for each action space. The controller – module shared between all spaces – is a transformer architecture (Vaswani et al., 2017a) (although any architecture that can handle variable-length inputs and tokens can be used). Since instructions and action descriptions (we use textual descriptions, such as e.g. "The

action space is cardinals.") are expressed in text, we can directly feed to the controller via simple word embedding layers.

The perception modules take in an observation and output a fixed-size embedding. The architecture for each perception module is chosen to best fit the modality of the corresponding observation space (the inventory, when represented as a list, is embedded with a 2D convolutional network). However, the number of vectors output by those specialized architectures vary from space to space (or even sample to sample for List space). In order to unify these outputs as input to the controller, we use adapter networks for each observation space. An adapter takes input of embedding of variable length and outputs embedding of fixed length, which is then input to the controller. That is achieved by using the network as cross-attention layers in enc-dec transformers.

The observation embedding is then concatenated with instruction and action space description embeddings. We also concatenate a fixed number of special padding tokens before feeding it into the controller. Among output vectors from the controller, we select the ones that correspond to the padding tokens, which gives us a fixed number of embeddings vectors to work with. Those embeddings are then fed into the action space specific action module, whose output corresponds to the dimensions of the corresponding action space. The fixed size of action embedding enables faster adaptation to new action spaces.

4.5 Related Work

Single Modality. Compositional generalization is often studied at the level of a single domain. In vision domain, models are tested if they can recognize an image that contains an unseen combination of different visual properties (e.g. shape, color), with emphasis on disentangled representation (Xu et al., 2022). Instruction and task is another domain where compositional generalization is well studied (Zhang et al., 2018; Zhou et al., 2022). At test time, the agent is given an unseen instruction or task that usually can be accomplished by chaining together already learned skills (Lake & Baroni, 2017). This idea of learning a set of skills that can be

composed together can be traced back to options framework (Sutton et al., 1999c) and other hierarchical RL methods (Sukhbaatar et al., 2018). Given the recent success of large language models, language is increasingly being used: Jiang et al. (2019) leverages natural language for hierarchical RL; Huang et al. (2022a) uses large language models to decompose tasks into smaller subtasks; Mezghani et al. (2023) has a single model for both policy and language reasoner. Unlike these, the focus our paper is composition generalization across of different modalities.

Modular Architectures for Multi-task Learning. Modular architectures can be viewed as composition of internal modules of the agent, and often studied together with multi-task generalization. PathNet (Fernando et al., 2017) uses genetic algorithm to select a subset of a neural network to be used for a specific task, showing positive transfer from one task to another. Rusu et al. (2016) grows a neural network by adding new modules with each new task, but allowing them to connect all previous modules. Continual learning is enabled by freezing previous modules, but still positive transfer is observed between different Atari games. Similarly, Gesmundo & Dean (2022) both grows and and selects subsets of the network. However those methods require training on the target environment, while our method enables zero-shot generalization to an unseen environment while utilizing a simple end-to-end training. LegoNN (Dalmia et al., 2022) is an encoder-decoder model with decoder modules that can be reused across machine translation and speech recognition tasks. Our approach for connecting perceptual modules to the controller recalls Alayrac et al. (2022), where the authors use cross-attention to connect a vision model to a text Transformer, and Jaegle et al. (2021) where this idea is discussed more generally.

Multi-Embodiment Continuous Control. Devin et al. (2017); Huang et al. (2020) used Graph Neural Networks (GNN) to build modular architecture that can control many different physical bodies. Furthermore Huang et al. (2020) shows such architectures are capable of zero-shot generalization to a new physical body. Our work, as Kurin et al. (2020), uses a Transformer in place of the GNN. The “action

spaces" in this work are analogous to the body morphologies in those. However, here, we study composable generalization not just to different action spaces, but to perceptual and task spaces as well.

Language Model as Controller and Planner via Text Interfaces. Several works have shown how a language model used as can be a nexus between modalities, and controller or planner for embodied agents. The general theme is to use text as glue, and the language model as a central processor. For example Socratic agents (Zeng et al., 2022) combines multiple pre-trained models from different domains to create a system that can solve unseen task involving a novel combination of domains. Similarly Huang et al. (2022c) deploy a pretrained LM as a robotic controller by augmenting it with additional models that can interpret visual scenes in language. In Ahn et al. (2022), the language model is used to score affordances based on a task description, and as a planner, following Huang et al. (2022a). In this work, rather than connecting modules via text, we use self-attention, allowing end-to-end learning.

Transformers in Behaviorally Cloned Generalist Agents. Our work is closely related to Reed et al. (2022) and Shridhar et al. (2023), where the authors showed that end-to-end Transformers can be effective controllers for embodied agents with multi-modal perception and/or actions. As in those works, we train via behavioral cloning. While Reed et al. (2022) tokenizes all inputs and treats the Transformer controller as a monolith, we allow passing gradients to perceptual or task-specific submodules. In this, we are similar to Shridhar et al. (2023), but rather than consider a fixed perceptual and action space as in that work, we show that our setup allows compositional generalization between perceptual, action and task spaces, and fast adaptation to new spaces.

4.6 Experiments

In the following set of experiments, we examine the compositional generalization properties of COIN agent. First, we examine the ability of COIN to generalize to unseen environment instances (O_m, A_n, I_k), where combinations seen during training are selected uniformly at random. Next, we examine the case where the samples held out from training are a selected group of particularly challenging instruction and observation spaces. Lastly, we test the ability of COIN to adapt to new, completely unseen observation spaces O_{new} through finetuning.

All the experiments use the compositional environments described in Section 4.3. For training, we use a dataset of 2,048 episodes with near-optimal trajectories $\{\tau^{(t)}_{(m,n,k)}\}_{t=1}^{2048}$ for each of the $240 = 6 \times 5 \times 8$ possible combinations of observation, action and instruction spaces (6, 5 and 8 options respectively); some of which will be held out from training. The near-optimal trajectories are generated using the A* algorithm or hand-engineered optimal policy. We evaluate the performance of the trained agent by measuring the rate of successful completion on both unseen and seen environment combinations. The task is considered completed if the agent reaches the goal defined by the instruction within the first 100 steps. When evaluating the trained agent on seen environment combinations, we measure the performance on that environment instance generated using a different random seed, i.e. the exact initial state and instruction are likely to differ from train time.

As an architecture for the perception modules, we use a pre-trained ResNet-18 network (He et al., 2015) for image spaces; for Grid and List spaces we use a 2D and 1D convolutional networks; for Text and Symbol spaces, we use 1D convolutions. The controller is a pre-trained Distilled-GPT-2 (Sanh et al., 2020), while each action module is a simple feed-forward network with the output corresponding to the dimensionality of the action space. The dimensions of observation embedding are 10×768, and the dimensions of action embedding are 4×768, where 768 is the dimension of GPT-2 token embedding. Each network is trained for 80 epochs. An illustration of the architecture can be seen in Figure A.1. More details about the architecture and the training procedure can be found in the Appendix A.2.

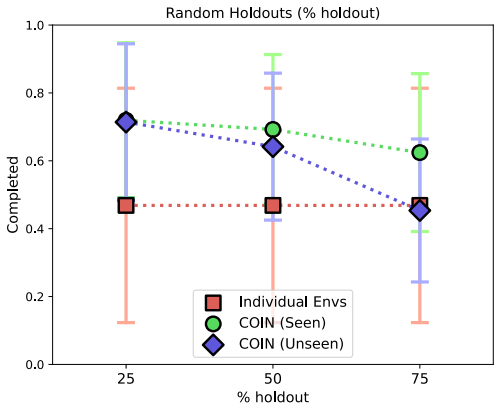


Figure 4.3: Agent performance at different percentages of held-out environments. Green: COIN agent on environments in the training data. Blue: COIN agent on environments *not* in the training data. Red: agent trained on only one environment.

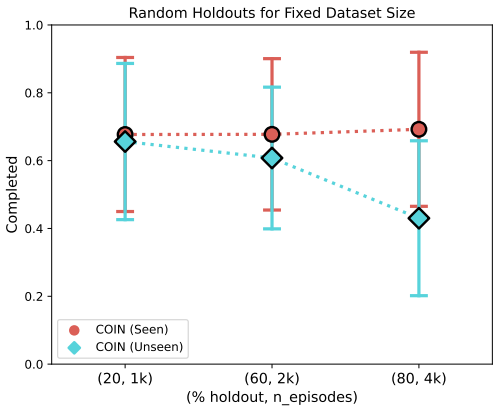


Figure 4.4: Agent performance at different percentages of held-out environments with a fixed size of training dataset. Red: COIN agent on environments in the training data. Cyan: COIN agent on environments *not* in the training data.

As a baseline, we use an agent trained on individual environment combinations using the same architecture, i.e. we train a separate agent on each of the 240 environment combinations. Note that in these cases, there is no weight sharing and each of such networks will contain only one perceptual and action module.

4.6.1 Random Holdouts

We start by examining the case where the environment instances (O_m, A_n, I_k) included in the dataset have been chosen randomly by chance. To ensure relatively uniform coverage of all spaces, the procedure for selecting environment instances guarantees that each space individually has been included in least four combinations. We vary the percentage of environment instances held out from the training: either 25, 50 or 75% of all possible combinations. We report separately the performance on the environments included in the training data (seen) or held out for training (unseen).

In Figure 4.5 we take a look at the performance difference for each of the spaces individually, i.e. we fix one of the spaces (observation, action or instruction) and average over the rest. We consider the case where 25% combinations are held out

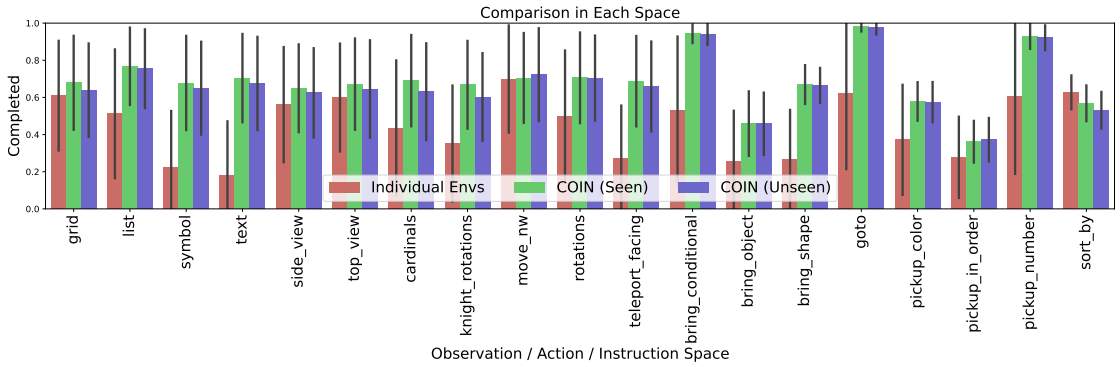


Figure 4.5: Comparison of performance between individual observation, action and instruction spaces. For each space, we report the performance averaged over all environment combinations containing that space (the error bars represent standard deviation). For trained on samples from 75% environment combinations, we report the completion rate on environment instances included (green) and *not* included (blue) in the training data. The performance of an agent trained on only one environment instance is shown in red.

(results with 50 and 75% held out combinations can be found in the Appendix A.3.1. Here we can see that COIN outperforms or matches the individual agents in all but one space (instruction space *Sort by Property*). We can also see that performance on unseen combinations matches the performance on seen combinations, which implies that the agent achieved near-perfect generalization to unseen compositions (with the remaining generalization gap being a consequence of either optimization difficulties or poor generalization to unseen observations and instructions). The greatest performance gains are seen on token observation spaces (*Text*, *Symbol*), which are particularly challenging for optimization and may particularly benefit from additional supervision provided by co-training on multiple observation spaces: the learning may be bootstrapped by learning a good controller on other, easier spaces.

The completion rates averaged over all the environment instances can be seen in Figure 4.3. From there, we can see that the COIN agent generalizes to unseen environment combinations extremely well, even outperforming the agents trained on individual environment combinations when the holdout rate is over 50%. The performance of COIN agent drops as we decrease the number of combinations included in the training data as expected. The error bars represent standard deviation over 240 environment instances.

Method	Completion Rate
Individual Envs	0.35 ± 0.18
Random Holdouts (25%)	0.34 ± 0.07
Hard Holdouts (8%)	0.26 ± 0.08
Hard Holdouts (32%)	0.21 ± 0.12
Hard Holdouts (55%)	0.20 ± 0.10

Figure 4.6: Agent performance on a set of 20 particularly challenging environment instances $\mathcal{E}_{hard}$. For COIN with both random and hard holdouts, we report zero-shot performance. In hard holdouts, the entire $\mathcal{E}_{hard}$ was held out from the training data; whereas in random holdouts, a random selection of 25% of combinations was held out. For hard holdouts, we report results where a total of 8, 32 and 55% of environment instances (including $\mathcal{E}_{hard}$), were held out. We also report the performance of agents trained on individual environments from $\mathcal{E}_{hard}$.

To evaluate the relative importance of using more data for each of the training environment combinations versus using more environment combinations in the training dataset, we run an experiment where the total number of episodes seen during training is kept constant while varying the holdout rate and number of episodes used in training. The total number of episodes used in training is always 192k, with the percentage of environment combinations used in training and number of episodes being: $(80\%, 2^{10})$, $(40\%, 2^{11})$ and $(20\%, 2^{12})$. The results can be seen in Figure 4.4. We find that for compositional generalization, it is more advantageous to use more environment combinations.

4.6.2 Hard Holdouts

Next, we consider the hard case where the holdout set is composed of particularly challenging combinations, either in terms of data collection or training time. We are particularly interested in this case, as in practice, there may be cases where data collection is much more challenging for some combinations (e.g. when some observation spaces correspond to data collected on real robots instead of data collected in simulation, where it can be hard to evaluate completion of some instructions outside of simulation). In these cases, it might be advantageous to collect the data on easier combinations for training and obtain good zero-shot performance on hard combinations without requiring data collection or training.

To construct hard combinations, we selected image observation spaces ($\mathcal{O}_{hard}$ = { *Top View*, *Side View* }) and two of the hardest instruction spaces ($\mathcal{I}_{hard}$ = { *Bring Object*, *Pickup In Order* }) as the performance on these spaces is generally the lowest, training trajectories are the longest and training on images takes more time. The hold-out set $\mathcal{E}_{hard}$ then consists of all combinations (O_m, A_n, I_k) where $O_m \in \mathcal{O}_{hard}$ and $I_k \in \mathcal{I}_{hard}$. In our case, this will be a total of 20 environment combinations. We train the COIN agent on the remaining 200 combinations, or a randomly sampled set of 75 and 50% of the remaining environments (in total, this corresponds to 8, 32 and 55% of all possible combinations being held out respectively). For hard holdouts, we report results on 5 different random seeds.

As shown in Table 4.6, we find that while not matching the performance of agents trained individually on those combinations or when the combinations are held out randomly, we still observe good transfer from easier to hard combinations, despite never seeing the particularly hard combination of observation and instruction in the training dataset.

4.6.3 New Perception Spaces

Lastly, we examine if COIN agent can effectively and efficiently incorporate new observation spaces. This is particularly relevant in a continual learning setting, where over a lifetime, new perceptual spaces may need to be added, without hurting the performance on spaces the agent has been already trained on or requiring training again from scratch on the entire dataset including the new observation spaces. Modular architectures have the potential to integrate new observational spaces without affecting the performance on other spaces by training only the new perceptual module, while freezing the controller and action modules. Moreover, training only the perception modules may require less data and converge more quickly.

To test this, for each observation space O_{new}, we first take out all the samples with that observation space from the training data (in total 40 different environment combination) and train on the data from randomly selected 75% of the environment

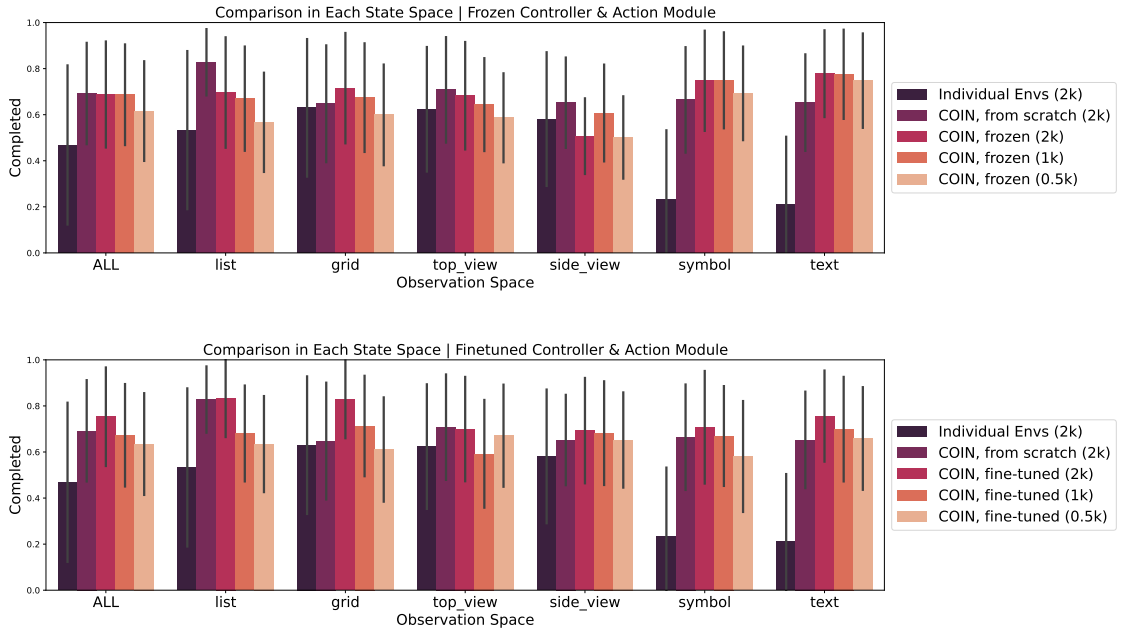


Figure 4.7: Performance of COIN agent on the 40 environment combinations $\mathcal{E}^O$ containing a newly added observation space O, for each of the six available observation spaces. The controller and action modules are trained on 75% of all randomly selected combinations *not* including $\mathcal{E}^O$. In the top figure, we only train the newly added perceptual module (i.e. without affecting the performance on other tasks), whereas in the bottom figure, we fine-tune the entire network. We report the results using 2048, 1024, or 516 episodes from each environment in $\mathcal{E}^O$ for training. We contrast these results to an agent trained from scratch on $\mathcal{E}^O$ and agents trained individually on each task in $\mathcal{E}^O$. The results are reported over 3 random seeds, with the error bar representing standard deviation over all environment instances in $\mathcal{E}^O$

combinations. Next, we take the trained COIN network and add the freshly initialized perception module for O_{new}, which is trained on the data from all the environment combinations containing O_{new}. The weights of the controller and action modules are kept constant. To compare the data requirements of adding the new perceptual spaces to already trained controller and action modules, to training the entire network from scratch, we train the new module using 2048, 1024, or 516 episodes from the dataset (full, half, or one-fourth of the dataset respectively). For comparison, we also try fine-tuning the entire network on the combinations with the new observation space (i.e. without freezing weights of the controller and action modules). We also compare the results to the modular network trained on the same 40 environments from scratch (i.e. without transfer from other observation spaces).

Results on the new observation spaces with freezing of the controller and action modules can be found in Figure 4.7 Top and without freezing in Figure 4.7 Bottom. We find that, when averaged over the observation spaces, by training only the new perception module, we can match the performance obtained by training from scratch and outperform training on individual environments. Moreover, we can match the training from scratch with one-fourth of the data. This is likely due to transfer from other tasks, where the new observation just needs to be mapped to a representation already understandable by the controller. We are finding that fine-tuning the entire network is not necessary for achieving good performance, hence a new observation space can be incorporated without affecting performance on other environment instances.

4.7 Conclusion

In this paper, we proposed a modular architecture with differentiable interfaces to various modalities of perception and action. These interfaces are connected to a shared controller, enabling passing gradients and end-to-end backpropagation, while supporting knowledge sharing. We developed a new environment in which perceptual modalities, sets of actions and types of instructions can be independently varied. This environment allowed us to systematically study compositional generalization across different modalities.

An agent trained with the modular architecture demonstrated zero-shot generalization when tested on unseen combination of modalities, outperforming an agent trained only on that combination. Furthermore, on a set of held-out combinations that were challenging to learn for a "single-environment" agent, the modular agent still showed zero-shot generalization. Lastly, we have shown that new perceptual modalities can be easily incorporated by training only the interface processing that modality. These results show that modular architectures can engender compositional generalization and cross-domain transfer without any special training scheme, paving a path toward a generalist agent capable of rapidly adapting to a new perception or control.

5

Diverse in Value-Relevant Features

5.1 Introduction

The automatic learning of useful skills is a fundamental challenge in scaling reinforcement learning (RL). Such behavioural abstraction can reduce the effective time horizon of RL problems and enable more coordinated exploration (Dietterich, 1998; Nachum et al., 2019). Furthermore, if skills are modular, they can facilitate transfer learning, as each skill constitutes a part of a solution (Andreas et al., 2017a; Frans et al., 2018; Igl et al., 2019b).

Rather than specify skills manually, which can be difficult or expensive, many existing approaches look to learn them automatically. In particular, one family of algorithms learns without use of a reward function, yielding skills that can be used across a variety of downstream tasks. These recent works (Gregor et al., 2016; Eysenbach et al., 2019; Sharma et al., 2020) learn distinct skills that are optimized such that trajectories generated by different skills are maximally distinguishable, according to the states visited by each skill. This yields a coordinated optimization problem: a discriminator network is optimized to distinguish skills, and skill policies are optimized to be distinct. However, skills learned in this way do not always capture meaningful structure in the environment, as in rich state spaces,

the discriminability objective may have many optima, leading to many different ways in which skills could differ.

In practice, without explicit encoding of prior knowledge about the specific features with respect to which skills should be diverse, the learned skills are often trivial. For example, agents may learn subtle manipulations of state that are imperceptible to humans, or subtly change joint angles in MuJoCo tasks (Achiam et al., 2018; Eysenbach et al., 2019). These skills, while perfectly distinguishable from each other, do not manipulate the state in ways useful for typical tasks of interest. This problem becomes acute when the state space is large and some features are easily controllable. As an extreme example, imagine a cooking agent that employs visual input as the state. In this setting, we cannot explicitly express the temperature or spiciness of a dish in pixel space. This prohibits the use of existing methods, as without access to hand-crafted features, the agent would learn to manipulate complex state spaces in easy-to-learn but unhelpful ways (e.g., knocking all the spices from the shelf, but adding none to the dish). Instead, if the agent specifically learns to manipulate features that determine the spiciness and temperature of the dish being prepared, then the agent will be more likely to learn to prepare different dishes.

To achieve this, in Smith et al. (2022), we propose DIVRS (Diverse In Value Relevant Space): a semi-supervised, sequential method that bridges the gap between hand specification and unsupervised learning of behavioural abstraction. Our framework uses learning to make it easier to specify the features of the environment in which skills should be differentiated. To do so, we first make use of a small amount of reward signal to learn a representation of state that enables prediction of task performance. Then, we learn skills that manipulate the space of those relevant and controllable features. We demonstrate that, compared to purely unsupervised counterparts, such skills are better able to cover the space of possible value functions that depend on those relevant features. Such skills can then be utilized on more challenging downstream tasks that share those same features.

5.2 Setting

We are given access to an environment $\langle S, A, T, P_0 \rangle$ that can be freely explored. The environment also consists of a set of tasks $\mathcal{M} = \{M_i\}_{i=1...N_m}$, with each task M_i specified by $\langle r_i, \gamma_i \rangle$. Our goal is to pre-train a set of skills $\pi(a|s, z)$, $z \in \{1, 2, \ldots, N_z\}$ such that any of the tasks M_i can be solved quickly. For example, when we anticipate having to train multiple agents on different tasks from $\mathcal{M}$, we can reduce overall training costs by pre-training a set of useful skills only once. During pre-training, we use a small subset of representative *source* tasks $\mathcal{M}_s \subset \mathcal{M}$.

In general, these tasks can be randomly sampled from the task distribution, or chosen by hand to be easy to solve, or represent well the space of tasks in $\mathcal{M}$ well. As discussed further in Section 5.3, access to $\mathcal{M}_s$ allows us to learn skills that are relevant for $\mathcal{M}$, without having to directly solve the large number of tasks in $\mathcal{M}$ individually, or solve particularly difficult tasks in $\mathcal{M}$. In fact, the source tasks do not even need to be fully solved; only effective policy evaluation is needed to learn useful skills. For this to be feasible, we assume the existence of a shared structure between the tasks. Specifically, we assume there is a set of low-dimensional shared state features $\psi(s)$ with $\dim(\psi(s)) << N_s$, that are useful for predicting the long-term performance on all of the tasks from $\mathcal{M}$, though in general, these features may be difficult to learn.

Such assumptions hold in many real-world problems. In navigation tasks, for example, the value of a state is primarily determined by the agent's location or its distance from particular objects (with the same feature potentially having positive or negative value depending on the task instance). For a cooking robot tasked with making many different dishes, the features relevant for differentiating the dishes include the amounts of ingredients, the temperature under which it was cooked, the spiciness or acidity of the dish, etc.

To find skills that are useful for navigation, a common approach has been limiting the observation space of the discriminator to the position of the agent's center of mass (referred to as an x-y prior) (Eysenbach et al., 2019; Sharma et al., 2020) or the difference between consecutive states (Achiam et al., 2018) to encourage skills that

exhibit movement. However, there are many tasks in which the relevant features of the state space are unknown a priori or are hard to specify in this manner.

5.3 Method

For sufficiently large state spaces in which the agent is able to easily control the state, diversity in the raw state space is not sufficient to learn skills that are relevant for downstream tasks. For example, the set of skills discovered by a MuJoCo Ant agent trained with DIAYN (Eysenbach et al., 2019) tend to take actions that affect easily-controllable features like joint positions and orientation. While these skills are distinguishable, they do not capture meaningful behaviours such as different walking gaits or even involve significant or consistent movement in the x-y plane, rendering them useless for most tasks, including navigation. The underlying problem is that some features of the state, like joint angles, are much more directly influenced by actions and thereby easier to control by the policy than others, like the location of the agent, which requires a longer sequence of coordinated actions to change.

To find skills that are useful for navigation, a common approach has been limiting the observation space of the discriminator to the position of the agent's centre of mass (referred to as an x-y prior) (Eysenbach et al., 2019; Sharma et al., 2020) or the difference between consecutive states (Achiam et al., 2018) to encourage skills that exhibit movement. However, there are many tasks in which the relevant features of the state space are unknown a priori or are hard to specify in this manner.

To address these problems, we propose DIVRS (Diverse In Value Relevant Space), which utilizes the source tasks in $\mathcal{M}_s$ to extract a feature mapping $\psi_\omega(s) : S \to S^\psi$ that captures only those variations in the state that matter for the tasks of interest. By then restricting the discriminator's observations to such a latent representation, we learn skills that are not only diverse, but diverse in relevant features. DIVRS consists of two phases of learning. The aim of the first phase is to learn a set of good state features, using the small set of source representative tasks $M_i \in \mathcal{M}_s$. After learning good features for the source tasks, we need to extract the representation $\psi_\omega(s)$ from this agent. In the second step, we learn skills that reach

states that are *diverse* in those features. The pseudo-code for both phases can be found in Algorithm 1, with the notation explained in the rest of this section.

Algorithm 1 DIVRS

Input: set of source tasks $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, coverage policy μ
Output: set of skills $\pi_\theta(a|s,z)$

Phase 1 - Learning State Representation $\psi_\omega(s)$

initialize $\psi_{\hat{\omega}}(s), \{w_1, ..., w_k\}, f_v$
while $\psi_{\hat{\omega}}(s)$ is not converged **do**
  sample $\mathcal{M}_i$ from $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$
  for j in n_steps **do**
    transition_batch $\leftarrow$ rollout_policy($\mathcal{M}_i$, μ)
    $\psi_{\hat{\omega}}(s), \{w_i\}, f_v \leftarrow$ **PEval**(transition_batch, $\psi_{\hat{\omega}}(s), \{w_i\}, f_v$)
$\psi_\omega(s) \leftarrow$ **Distillation**($\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, $\psi_{\hat{\omega}}(s), \mu$)

Phase 2 - Learning Skills $\pi_\theta(a|s,z)$

fix the parameters ω of $\psi_\omega(s)$
initialize $\pi_\theta(a|s,z), q_\phi(z|\psi_\omega(s)), Q_\nu(s,a|z)$
while not converged **do**
  sample $z \sim P(z)$
  for j in n_steps **do**
    exps $\leftarrow$ rollout_policy($\pi_\theta(a|s,z)$)
    exps.reward $\leftarrow \log q_\phi(z|\psi_\omega(s_t)) - \log P(z)$
    $\pi_\theta(a|s,z), Q_\nu(s,a|z) \leftarrow$ SAC_update(exps, $\pi_\theta(a|s,z), Q_\nu(s,a|z)$)
    $q_\phi(z|\psi_\omega(s)) \leftarrow$ optimize_dscr_loss(exps, $q_\phi(z|\psi_\omega(s))$)

Algorithm 2 PEval

Input: transition batch D, shared embedding $\psi_{\hat{\omega}}(s)$, task embedding w, value network f_v

$S, A, R, S' \leftarrow D$
$\delta \leftarrow (f(\psi_{\hat{\omega}}(S), w) - R - \gamma f_v(\psi_{\hat{\omega}}(S'), w))$
$v \leftarrow v - \delta \nabla_v f_v(\psi_{\hat{\omega}}(S), w)$
$\hat{\omega} \leftarrow \hat{\omega} - \delta \nabla_{\hat{\omega}} f_v(\psi_{\hat{\omega}}(S), w)$
$w \leftarrow w - \delta \nabla_w f_v(\psi_{\hat{\omega}}(S), w)$
return $\psi_{\hat{\omega}}(s)$, w, f_v

Algorithm 3 Distillation

Require: A set of source tasks $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$, target embedding $\psi_{\hat{\omega}}(S)$, coverage policy μ
Output: Distilled Embedding $\psi_\omega(S)$

initialize bottleneck network ω, reconstruction network d_z
while $\|\psi_{\hat{\omega}}(s) - d_z(\psi_\omega(s))\| \geq \epsilon$ **do**
  sample $\mathcal{M}_i$ from $\{\mathcal{M}_1, ..., \mathcal{M}_k\}$
  for j in n_steps **do**
    $\leftarrow$ rollout_policy($\mathcal{M}_i$, μ)
    $l \leftarrow$ RegressionLoss($d_z(\psi_\omega(S)), \psi_{\hat{\omega}}(S)$)
    $z \leftarrow z - \nabla_z l$
    $\omega \leftarrow \omega - \nabla_\omega l$
return $\psi_\omega(s)$

5.3.1 Phase One

Choice of features In order to capture features that are relevant specifically on the tasks of interest, we propose that $\psi_\omega(s)$ be given by features of the value function that are shared across tasks. Since the value function captures the long-term performance of the agent, the features that it depends on describe exactly how successful the agent is likely to be—if the agent is seeing features that predict high values, it is likely to perform well on the given task. By using the state features of the value function, we can obtain skills that manipulate ψ_ω in a way that maximizes a wide range of value functions in $\{\mathcal{M}_i\}$. We choose features of the value function

over a learned reward model because the value function is smooth compared to the reward function and carries more information about the long term effect of various features of the environment. Note that, while there are many other approaches one could use to learn features in RL, including autoencoders (Lange & Riedmiller, 2010), and successor features (Schaul et al., 2015), these methods generally do not depend on task performance, which is our focus here.

Learning features To learn value-relevant features, we fix a coverage policy μ across all tasks for the first phase and simply perform policy evaluation, while ensuring that $\psi_\omega(s)$ is optimized across all tasks. This policy could be random, derived from expert behaviours, or chosen to cover state space uniformly. Alternatively, the policy can be conditioned on the active task, and optimised alongside the value function via policy improvement. In our experiments, random policies were sufficient to derive good skills. Due to the variance inherent in evaluation of random policies, we found that it was easier to first train a value network with a high dimensional embedding, $\psi_{\hat{\omega}}(s)$, which is later compressed in a distillation phase, which we describe below. The training procedure for $\psi_{\hat{\omega}}(s)$ is detailed in Algorithm 1 (Phase 1), and Algorithm 2. Since DIVRS primarily relies on policy evaluation, optimal policies do not need to be learned for the source tasks. Instead, the value function only needs to capture the features that are predictive of future rewards.

Value function architecture Crucially, for DIVRS to work, the *reward on the source tasks does not have to be dense* as long as policy evaluation can be performed with good coverage of states that lead to rewards. To ensure that $\psi_{\hat{\omega}}$ remains task-independent and is used across all tasks, we can choose our class of value function estimators V_i^μ (as in (Schaul et al., 2015)) such that values can be factored as a combination of a state-independent task embedding and the learned features, i.e. $V_i^\mu(s) \approx f(\psi_{\omega(\hat{s})}, w_i)$, where $f(\cdot)$ is a simple function like inner product or a small neural network, μ is a fixed (potentially task-conditioned) policy, and w_i is the state-independent embedding of task i.

Compression of features DIVRS relies on finding a compressed representation ψ_ω of the state. If the representation is insufficiently compressed, i.e., contains too many random projections from irrelevant features of the environment; or if the representation is insufficiently general, i.e., captures spurious but controllable features of the environment (such as changes in the background relative to the agent), the skills that learn to control the resulting features may not be useful. Hence regularization and compression play a key role in making this approach work.

Information bottleneck methods (Tishby et al., 2000) encourage learning representations that are both general and do not contain projections from irrelevant parts of the state space. These methods capture the notion of a minimal sufficient statistic by maximizing the information conserved about the relevant prediction (in our case the value function), while maximizing the amount of information lost between the raw state and the encoded state: $\min_\omega I(s; \psi_\omega(s) - I(\psi_\omega(s); V_i^\pi(s)))$. This can be done explicitly (Alemi et al., 2016; Igl et al., 2019a), though there is some evidence that it occurs naturally in capacity constrained deep networks (Tishby & Zaslavsky, 2015; Shwartz-Ziv & Tishby, 2017), which we found give sufficiently terse encodings. To promote this further, we make use of regularisation during training (Cobbe et al., 2019) and, for the second phase, take the compressed, ψ_ω from a value function that is distilled after learning (Rusu et al., 2015), in order to capture features corresponding to a minimal sufficient statistic for value.

The pseudo-code for this part of the method, can be found in Algorithm 3. ψ_ω is trained through a regression objective to contain minimal information to predict the outputs of $\psi_{\hat{\omega}}$, despite being a lower dimension embedding. This is accomplished by training a reconstruction network d_z that takes the compressed representation ψ_ω and projects it into the original representation space.

5.3.2 Phase Two

Learning skills In the second phase, without utilizing rewards, we train a set of skills using VOD methods. Instead of providing the discriminator directly with prior knowledge about the space in which the skills should be diverse such as an

x-y prior—which may not be accessible in settings environments where the skills need to be learned—we instead employ the compressed features $\tilde{\psi}_\omega$ learned in the first phase. While our approach is compatible with any VOD method, in our experiments we employ either forward or reverse MI forms of DIAYN (Eysenbach et al., 2019; Campos et al., 2020). The discriminator and skills are trained in parallel, with the discriminator trained to minimize the prediction loss at each state independently–i.e., $\mathcal{L}_\tau(\phi) = \sum_{t\in\{1..T\}} \mathcal{L}_t(\phi)$, where $\mathcal{L}_t(\phi) = -\log q_\phi(t)$. The discriminator loss at a timestep t is $q_\phi(t) = q_\phi(\psi_\omega(s_t)|z_t)$ for the forward MI objective, and $q_\phi(t) = q_\phi(z_t|\psi_\omega(s_t))$ for reverse MI. The resulting pseudo-reward for the skill is $r_t = \log q_\phi(t) - \log Z_t$ with normalisation term $Z_t = \sum_z q_\phi(\psi_\omega(s_t)|z)P(z)$ for the forward objective and $Z_t = P(z_t)$, with uniform skill prior distribution $P(z)$. The parameters of the state embedding $\psi(s)$ are fixed for the duration of this phase. For the description of forward form objective, see Section 2.2.3.

Skill use DIVRS results in a set of skill policies $\pi_\theta(a|s,z)$ parametrized through z. These policies are learned to visit distinct regions of the learned state space, learning to control exactly the features relevant to the value of task distribution. This means that these skills can be used on unseen downstream tasks $\mathcal{M}$ with the shared structure, either by selecting and fine-tuning the best performing skills or by training a meta-controller $\pi(z|s)$ as in Eysenbach et al. (2019).

5.4 Related Work

Unsupervised Skill Discovery. Several methods have been developed for the automatic discovery of skills in RL. Particularly relevant, due to their unsupervised nature, is a category of information-theoretically motivated, reward-agnostic methods, which optimise policies according to various criteria, centered around notions of discriminability (Gregor et al., 2016; Eysenbach et al., 2019; Achiam et al., 2018; Sharma et al., 2020). These techniques are generally highly underconstrained, in that many sets of skills satisfy the optimization objective, meaning that they produce uninteresting or useless skills in practice, unless significant task-specific guidance

is provided (Achiam et al., 2018). Hausman et al. (2018) combine discriminability and task rewards; however, they aim to find multiple solutions to the specified task.

Transfer Learning. Several prior works also learn skills from a set of tasks (Thrun & Schwartz, 1995; Pickett & Barto, 2002). Frans et al. (2018) extract skills from a set of tasks, but unlike our approach, are restricted to downstream tasks that can be solved by a composition of the source task policies. Igl et al. (2019b) relax this restriction by also learning termination functions, allowing the composition of sub-policies from the source tasks. More widely, KL-regularization can be used to transfer knowledge between tasks Teh et al. (2017); Galashov et al. (2018); Tirumala et al. (2019), by using skill hierarchies to implement various inductive biases. Without the complications that can arise by learning temporal abstractions, Wulfmeier et al. (2019) show that sharing policies across closely related tasks can speed up training. However, all of these approaches are restricted to policies that are the same or close (as measured by a KL-divergence) to those on the source tasks. By contrast, we learn policies that are as diverse as possible, while using source tasks only to identify the relevant part of the state space.

Hierarchical Reinforcement Learning. Skills are often learned as part of a hierarchical policy. To learn a useful and diverse set of skills, one can rely on specified subgoals for each skill (Sutton et al., 1999d; Kulkarni et al., 2016), which can be found analyzing the structure of the MDP (McGovern & Barto, 2001; Şimşek & Barto, 2009), previous policies (Goel & Huber, 2003; Tessler et al., 2017) or predictability (Harutyunyan et al., 2019). Predictability, in contrast to discriminability, is the notion that skills should *terminate* in a predictable state, instead of *behave* in a way that is different from other skills. Subgoals can also be generated by a higher-level policy, either set in (Nachum et al., 2018a) or in a latent space with learned semantics (Vezhnevets et al., 2017; Nachum et al., 2018b).

Representation Learning. Our work is also related to representation learning, in particular the learning of state-abstractions (Abel et al., 2020; Li et al., 2006; Abel et al., 2016; Jong & Stone, 2005; Igl et al., 2019a). However, unlike this prior work, our state-abstraction is only value-relevant, not action-relevant. In other words, our compressed state-representation should contain information with which progress on tasks can be measured, but *doesn't* necessarily contain information which allows for progress on tasks. For example, while in the ant-task, the x-y location is sufficient to approximately determine the value-function, an optimal policy would also require the location and orientation of various joints - exactly the type of information we aim to remove from our representation. Similarly, while (Abel et al., 2019) also utilizes an information theoretic trade-off between compression and quality of the representation, their performance evaluation captures the quality of the policy, while we only capture the quality of value predictions. Silver et al. (2017) and Oh et al. (2017a) also learn a state-abstraction, but aim to capture even more information in the latent space, by also predicting temporal dynamics. On the other hand, Liu et al. (2020) and Goyal et al. (2019a) capture only the task-goal in the latent space, not information relevant to measuring progress w.r.t this goal. Similar to our work, Sermanet et al. (2018) use a learned representation to generated policy rewards. However, unlike our work, they utilize a metric-learning loss to learn the representation and apply their method towards viewpoint independence, not skill discovery.

Other. Similar settings with factored reward functions $r_w(s,a) = \phi(s,a)^T w$ have been studied (Barreto et al., 2017; Borsa et al., 2019), although, there is less emphasis on learning a compressed state representation as it does not benefit from lower dimensionality of $\phi(s)$. In addition, it is typically required that the weights w of downstream tasks are learned or known in advance. The assumption that under certain smoothness conditions, optimal value functions can be factored as $V^*(s,g) = \psi(s)^T w(g)$ is also employed by Schaul et al. (2015).

5.5 Experiments

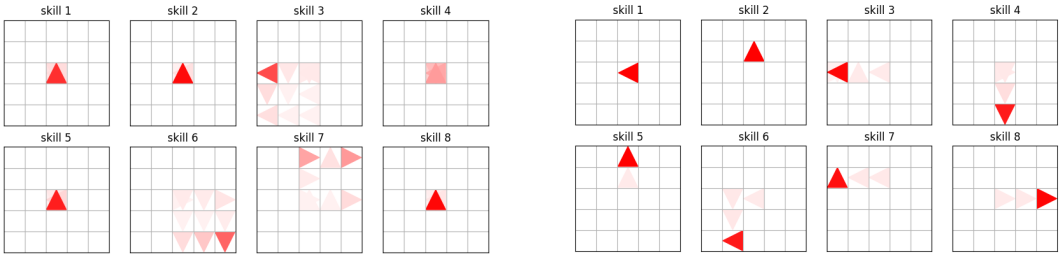
In this section, we examine the empirical results of our proposed method. Our aim is to constrain the space in which the skills are diverse and thus obtain qualitatively and quantitatively better skills.

5.5.1 Gridworld Experiments

We start by testing the proposed approach on a modification of a simple 7×7 gridworld from Chevalier-Boisvert et al. (2018) that has been expanded with extra controllable dimensions. In addition to rotating and moving forward, the agent can increase or decrease the value of one of the four added *shadow* dimensions, depending on which of the four cardinal directions it is currently facing. We can interpret them as, e.g., controlling brightness of the wall the agent is currently facing. The purpose of these added dimensions is to demonstrate the effect of expanding the space that the agent can control. Analogous to limb positions in Ant MuJoCo, they are not useful for value functions of navigation tasks, but are easier to control than the agent's position.

To learn which state features are relevant for navigation tasks, we train an entropy-regularized PPO agent (Schulman et al., 2017) on two simple navigation tasks, which involve navigating to one of the two corners of the room. The reward is received only upon task completion. During RL training, we initialize the agent at a random state to ensure generalization of the value function. The grid observation is embedded with convolutional layers, while the non-location observations – direction and added dimensions – are concatenated and embedded with fully connected layers. The two resulting embeddings are concatenated before being passed down to actor and critic heads. The task index identifies which of the two tasks the agent is in and is also embedded with a fully connected network and concatenated with the rest of the embeddings. In these experiments, we found that the embedding distillation step was not needed to achieve a parsimonious representation.

For unsupervised training, we fix the resulting convolutional and fully connected embeddings, and use them as features of the discriminator. For these experiments,

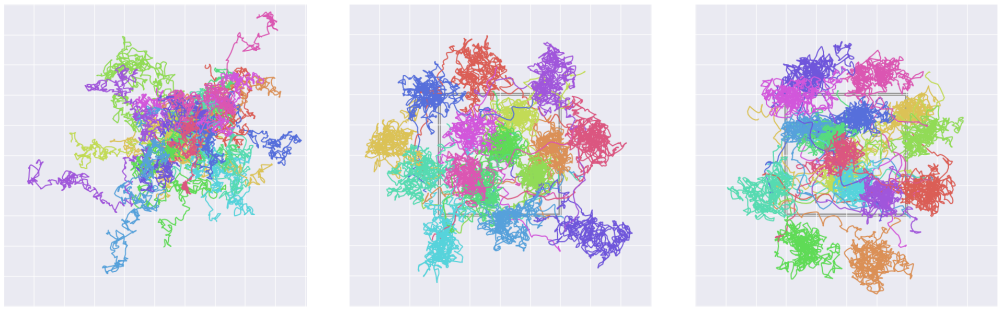


(a) DIAYN **(b)** DIVRS

Figure 5.1: Visualizing state visitation in the subspace of interest under 16 different skills trained using DIAYN (Eysenbach et al., 2019) (a) and DIVRS(b). Color intensity of the agent indicates the probability of an agent occupying given direction and position during an episode. Note the differences between the consistency of skills and which features differentiate skills.

we employed the reverse MI form of the DIAYN objective. The discriminator is parametrized as a single hidden layer neural network on top of those features. The skills architecture is similar to the policy in the first phase with the addition of a one-hot encoding of skill identity, which is embedded with fully connected layers and concatenated with the location and non-location embeddings. In experiments with DIAYN, we use the same architecture for the discriminator and skills.

Figure 5.1 visualizes which states are visited and with what consistency between different skill roll-outs. With the added shadow features, which are easily controlled, most DIAYN skills never leave the initial position (3,3), and instead learn to differentiate by controlling the shadow features or agent orientation. Skills that move in the grid do not reliably reach a particular position. By pre-training on two navigation tasks, we obtain a representation that ignores added dimensions of the state space as they do not affect the value function. In this case, this leads to differentiation of skills based on the position and orientation of the agent. Fig. 5.1a shows that five of the eight skills learned with DIAYN do not leave the initial position, and four of them seem to favour the initial orientation as well, instead manipulating the additional shadow dimensions. Those that do move do not do so consistently. By contrast, skills learned with DIVRS move consistently to specific and distinguishable states, both in terms of grid position and orientation.



(a) Reverse MI

(b) Forward MI with $x-y$-prior

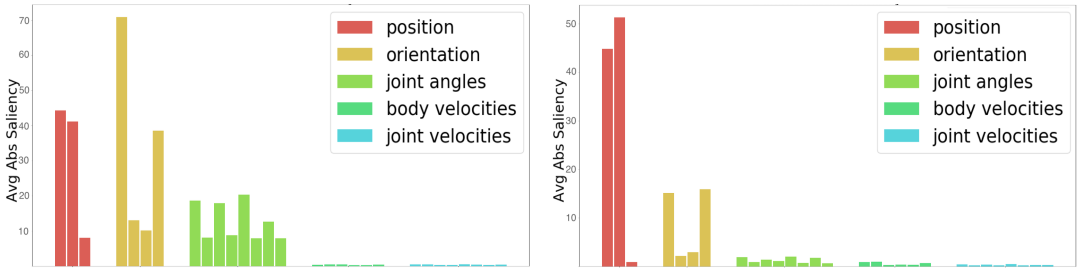
(c) Forward MI with DIVRS

Figure 5.2: Comparison of trajectory traces in x-y plane of 16 different Ant skills: (a) reverse MI, (b) forward MI with x-y prior, (c) forward MI with DIVRS. The grey square is for scale, denoting a 2m×2m area. Lines with the same color denote different rollouts of the same skill. Hand-crafting features (b) or transfer (c) were necessary to obtain skills that are consistent in and distinguished by visitation of different areas of $x-y$ plane.

5.5.2 MuJoCo Experiments

We start by examining skills trained on MuJoCo Ant (Todorov et al., 2012; Brockman et al., 2016), which has previously required hard-coded restrictions on the observation space of the discriminator (an x-y prior) to obtain skills useful for complex navigation tasks (Eysenbach et al., 2019; Sharma et al., 2020). In this way, the Ant environment is a good example of a failure mode for DIAYN that DIVRS is perfectly suited to solve. As source tasks, we use three simple navigational tasks. In each, the agent needs to reach a specific goal g, chosen from three fixed goal positions. The agent receives a sparse reward and the episode terminates when the agent is sufficiently close to the goal. We found that in MuJoCo tasks we tried, quality of skills obtained using the value features learned by policy evaluation of a random policy was as good as quality of skills obtained from value features of optimal policies, hence the results reported here use the latter. The value of a state in these tasks is determined mostly by the distance of the agent's center of mass to the goal; hence a representation that contains information needed for all three value functions needs to capture information about the position of the agent.

If the first phase of training, we use PPO (Schulman et al., 2017) to learn a goal-conditional value function $V(s, g)$. The value function is parametrized as $V(s, g) = f_v\Big(\psi(s) \odot w(g)\Big)$, where the function $f_v(\cdot)$ receives an element-wise



(a) DIAYN **(b)** DIVRS

Figure 5.3: Analysis of features used to differentiate skills by visualizing discriminator saliency. The features are grouped into positions, orientation, joint angles, body velocities, and joint velocities. Position features are much more salient for prediction with DIVRS (b) than DIAYN (a).

multiplication of the state and goal embeddings ($\psi(s)$ and $w(g)$ respectively). To ensure sufficient generality and feature compression, we distill the embedding $\psi(s)$, before passing it to the discriminator in the second phase. Distillation is performed on data gathered from a random policy. A network with fewer parameters is trained to predict the output of ψ through a narrow bottleneck layer $\bar{\psi}$. This bottleneck layer is then used as state input to the discriminator in the following phase.

In the second phase of training, we use SAC Haarnoja et al. (2018) with the forward MI objective to learn skills. We chose forward MI in order to ensure skills behaved consistently across the state space (see Appendix B.1.1 for a more detailed discussion) and SAC as it was the algorithm of choice for IT skill discovery in related work (Eysenbach et al., 2019; Sharma et al., 2020). In forward MI experiments, we model the skill-conditional state distribution $q_\phi(s|z)$ with a multivariate Gaussian in the learned space (or predefined space when using $x-y$ prior) with fixed uniform diagonal covariance and learned location parameter. The scale of variance was tuned as a hyperparameter, though it could in principle be optimized via the discriminator objective alongside the mean. More details about the experiment design can be found in Appendix B.1.2.

Qualitative Skill Evaluation

To analyze learned skills, we look at the x-y traces and saliency maps (Simonyan & Zisserman, 2014) of the trained discriminator. These x-y traces show how well the

(a) Bottleneck size: 4 **(b)** Bottleneck size: 16 **(c)** Bottleneck size: 64

Figure 5.4: Saliency maps of distilled embedding as a function of bottleneck size. We see that a small bottleneck is needed to fully compress the learned representation, removing all projections from irrelevant features.

skills cover the x-y plane and if they consistently reach certain areas. Saliency maps of the discriminator 5.3 show which features of the state are used to discriminate skills. Figure 5.2 compares x-y traces of vanilla DIAYN skills, skills of DIAYN with an x-y prior, and skills obtained with DIVRS. As reported by Eysenbach et al. (2019); Sharma et al. (2020), baseline skills trained with DIAYN do not venture far from the initial state position, whereas both DIAYN with an x-y prior and DIVRS explore the x-y plane well, though DIVRS is able to do so using learned, rather than explicit encoding of these relevant features. Note that we do not show skills trained on Forward MI without the x-y prior, as we found the high dimensionality of state space in this case to be too prohibitive for learning diverse skills. Figure 5.3 shows that vanilla DIAYN skills learn to be primarily differentiated via easy-to-control features like joint angles. Providing the discriminator a representation pre-trained on the source tasks limits the discriminator to relying on agent position. For further insight into the qualitative nature of the skills learned by DIVRS, we performed a comparison across VOD methods in Appendix B.1.1.

Degree of Compression Tests

In order to examine the effect of distillation on compression of the learned embedding, we conduct an experiment where we vary the width of the bottleneck across distilled models. Identical seeding (and thus data, since distillation is performed using a random policy) was used across distillation instances. As a proxy for quality of

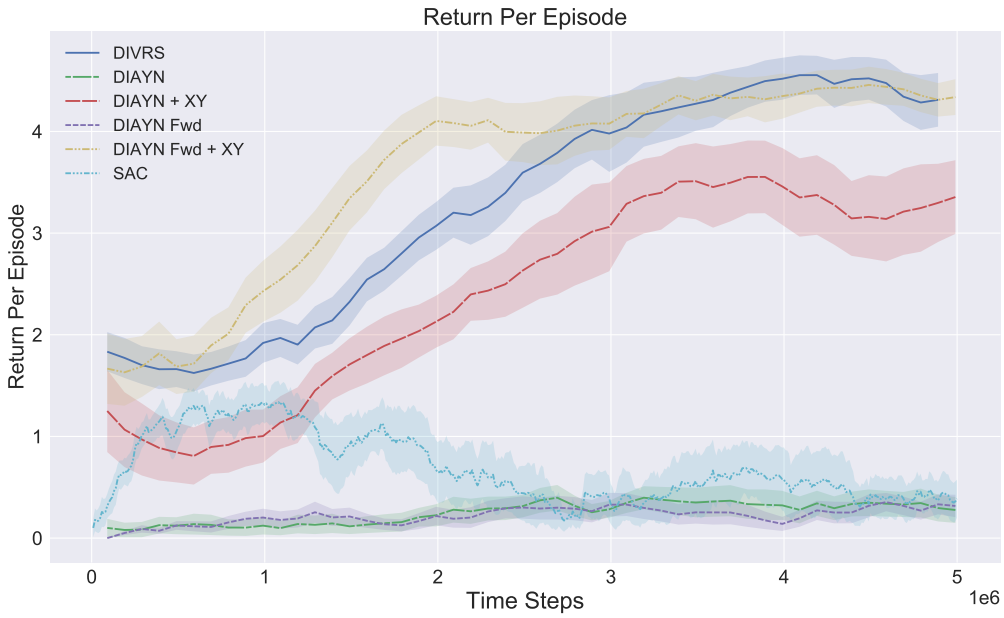


Figure 5.5: Performance on sparse sequential task: comparison of SAC and HRL agents trained on top of skills obtained with either DIVRS or DIAYN (with or without hand-crafted features)

compression, we examine the saliency maps of the distilled value function. More peaked saliency maps, where few features are highly salient and most are not, correspond to well-compressed embeddings. These saliency maps can be found in fig. 5.4. We find that as the bottleneck gets smaller, the number of projections from irrelevant features becomes fewer and less significant. This demonstrates the need for the distillation phase, as it further compresses the learned feature space.

Sparse Downstream Task with HRL

Next, we evaluate the effectiveness of skills learned by DIVRS on a challenging task that is difficult to solve without skills. In particular, we apply them to the sequential sparse navigation task from (Eysenbach et al., 2019). This task is challenging to solve even with skills if the space in which exploration is needed has not been specified. The agent starts at $(0, 0)$ and receives a sparse reward for reaching the corners of an axis-aligned square of length four. These points must be reached in a fixed order held constant across all episodes.

We employ a variant of SAC designed to work with discrete action spaces (Christodoulou, 2019) in the Semi-Markov Decision Process (SMDP) (Puterman,

Method	Avg Return
DIVRS	0.954 ± 0.031
DIAYN	0.360 ± 0.084
DIAYN + x-y-prior	0.972 ± 0.012
Forward MI DIAYN	0.384 ± 0.169
Forward MI DIAYN + x-y-prior	0.947 ± 0.040
SAC	0.174 ± 0.051

Figure 5.6: Few-shot performance on unseen goals: comparison of skills obtained with either DIVRS or DIAYN (with or without hand-crafted features)

2014) induced by our skills over the task MDP. Skills, which were executed for 1000 steps during training, are now terminated after (fewer) fixed time steps, before control is returned to our hierarchical agent. This skill duration was tuned as a hyperparameter across independent rollouts of the HRL learning algorithm. In addition to the state space provided to the skills, in order to ensure that states remain Markov, the skill coordinator policy receives a one-hot representation of the number of goals reached. We train both the flat and hierarchical policies for five million primitive time steps, although hierarchical methods trained at the SMDP level can only make use of data gathered at the hierarchical decision points.

Figure 5.5 shows the results of this experiment. The skills learned by DIVRS consistently achieve the maximal reward. This is comparable and even surpasses performance on the task with skills learned through explicitly specifying the space in which exploration is useful, though we have only specified a binary reward signal. We see that skills learned without any specification are generally unsuccessful in the task, as they do not explore the relevant feature space efficiently. Similarly, directly training flat models is unsuccessful in achieving consistent returns. For more details on the hyperparameters used in this experiment, see Appendix B.1.

Few-Shot Task with Skill Selection

In this section, we evaluate the zero-shot performance of learned skills on a sparse reward goal-seeking locomotion task. Goals are sampled from within the 4×4 box centred at the origin, and remain fixed for the evaluation period. We report

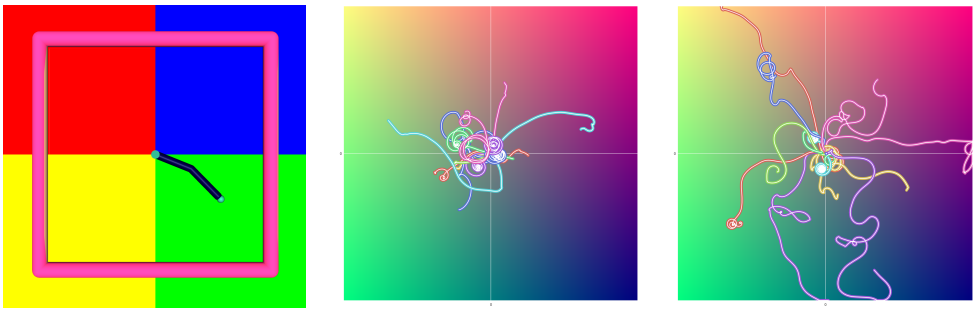
performance averaged over ten different goals sampled from this distribution. A reward of 1 is given when the agent is within $1m$ of the goal and the episode is terminated. Rewards are zero otherwise. The DIVRS skills are compared to DIAYN and x-y prior DIAYN skills for both forward and reverse modes, as well as a randomly initialized policy. Like Eysenbach et al. (2019), we evaluate all skills and choose the best one as a solution to the task. To account for the additional environment steps needed to evaluate all 16 trained skills, we fine tune the random policy on each task using SAC with the corresponding number of environmental interactions.

The results are shown in Table 5.6. As DIAYN skills do not cover much of the x-y space, they infrequently end up close to the goal. The random policy is able to reach the new goals occasionally, but the sparse learning problem is challenging, and thus the SAC agent is not able to learn with the little data available. By contrast, by covering the x-y space well, skill sets learned with DIVRS—and those learned using DIAYN with the x-y prior—are able to reach states in proximity to most sampled goals, leading to consistent rewards.

5.5.3 Pixel MuJoCo Experiments

In the tasks considered above, using hand-engineered features is feasible, as the relevant features are trivially selected from the available state features. To evaluate DIVRS in a more challenging setting, we consider a task where the relationship between state and relevant features is more complex because the observation space consists of images. We run our experiments on a variant of the MuJoCo Reacher with pixel observations, loosely inspired by the cooking robot example in Section 5.1.

The environment In addition to the original Reacher state, the agent must learn to manipulate an additional state subspace, given by the plane $[0, 1] \times [0, 1]$. We can visualise this additional subspace as a colour plane in RGB colour space, with a fixed blue chromaticity, and green and red chromaticities varying proportionally to the position in the plane. See the background of the trace plots in Figure 5.7 (b, c) for a rendering of this space. We colour the tip and the base of the Reacher



(a) Pixel Reacher Environment

(b) Skills Learned with DIAYN

(c) Skills Learned with DIVRS

Figure 5.7: Results on Pixel Reacher environment. The environment is illustrated in (a): the color of Reacher's tip and base is modified towards one of the four colors, with the colour determined by quadrant currently visited by the tip. The trajectories in the colour space of eight skills obtained with DIAYN and DIVRS are shown in (b) and (c) respectively. Lines with the same colour denote different rollouts of the same skill. DIVRS improves both coverage and consistency of skills in the subspace relevant for performance. Best viewed in colour.

agent according to its current position on this plane, so that the position in colour space is observable in the agent's (pixel) state space. We can imagine this colour space as an abstraction of flavour space for the cooking robot.

Depending on the position of the Reacher tip in the original state space, the colour of the tip and base change accordingly (this heavily abstracts the cooking process of moving around the kitchen, picking up spices, etc). As the Reacher tip moves further from the origin, the flavour changes more quickly, in accordance with the Euclidean norm of the tip position. To make the environment even more challenging, the direction of movement in colour space corresponds to the position of the tip (which falls in $[-1, 1] \times [-1, 1]$), randomly rotated by a multiple of 90° each episode. This rotation is represented in pixel space with four coloured panels below the agent, one in each quadrant, where each panel corresponds to movement in that direction in colour space (see Figure 5.7 (a)).

A visualisation of the environment is shown in Figure 5.7a. As in Hafner et al. (2019), the observations have been down-scaled to a 64×64 resolution, though here we use three-channel colour pixels. We concatenate two consecutive states to capture the joint velocities of the agent.

Learning Skills As before, the source tasks involve navigation to one of three fixed goals. However, now the goals exist in colour space, corresponding to a few sample dishes that the agent can be trained to learn. The reward on each of these source tasks is dense, corresponding to the negative distance from the agent's colour state to the goal. We use the same learning algorithm and the value function parametrization as in the MuJoCo Ant experiments; the main difference is that now only eight skills are learned, and the pixel observations are embedded with a convolutional neural network. More details on the network architecture and experiment design can be found in the Appendix B.1.3

To analyze learned skills, we examine the x-y traces in colour space, across two rollouts of each skill. The results are shown in Figure 5.7. For each method, we select the set of skills with the highest mutual information between skills and (embedded) states as measured on test rollouts. Since the observation space is rich, skills obtained through naive IT objectives can learn to be differentiated through many features, including body positions and velocities. In fact, in colour space, the DIAYN agent has reduced consistency of skills, and reduced state coverage compared to a random agent. By contrast, skills learned with DIVRS are much more consistent and are primarily differentiated by the agent's position in colour space.

5.6 Discussion

One of the limitations of our approach is that the user has to select a set of source tasks, or when available, randomly sample from the distribution of tasks of interest. Selection of source tasks is already a common practice in transfer learning literature, however, in some cases it may not be clear what the related source tasks are or the reward in source tasks will not be encountered sufficiently often to learn value-relevant features. Additionally, our two-phase approach is not well suited to settings where the task distribution changes over time. This could be approached in future work by iterating between phases of skill learning and skill usage, or by building hierarchies of skills, again alternating between skill construction and data collection. When the relevant features for the diversity objective are known and can be easily

specified, DIVRS is not an appropriate method to use, as the additional burden of learning good features is unnecessary. Furthermore, tuning of hyperparameters can be a challenge: parameters in later learning phases can be tuned independently, but hyperparameters in early phases need to be tuned in tandem with downstream hyperparameters, adding to optimisational complexity. Our method also relies on good exploration in both phases of the algorithm to ensure good state coverage. This is an important but mostly orthogonal problem to the one our method is solving. In our experiments, using a random policy was sufficient to ensure good state coverage, but more generally, methods such as marginal state matching (Lee et al., 2019) or pseudo-counts Bellemare et al. (2016) can be used to ensure enough exploration.

5.7 Conclusion

In this work, we address one of the core limitations of model diversity-based skill discovery methods in complex domains, which is that naive diversity objectives can be trivially satisfied by skills that offer little utility on downstream tasks. We addressed this limitation by developing a representation-learning approach to the automatic construction of sets of skills, and in doing so, enabling agents to more efficiently traverse the space of features indicative of performance on the target tasks. The primary contribution is the introduction of pre-training phase that obtains a new state representation for the discriminator, capturing state features that are more relevant for the tasks of interest. Our empirical results demonstrate that, when compared to flat baselines and skills learned without such pre-training phase, the skills constructed with our method provide better coverage of the relevant state space and lead to high performance on challenging sparse reward downstream tasks.

6 Meta-Gradients in Non-Stationary Environments

6.1 Introduction

Meta-gradient approaches to learning adaptive optimizers are a promising complement gradient-based optimizers in reinforcement learning (RL). By adapting relevant optimization hyperparameters or the entire update rule to the current domain, they often outperform well-tuned gradient-based optimizers (Schraudolph, 1999; Mahmood et al., 2012; Andrychowicz et al., 2016; Xu et al., 2018). The adaptability of meta-learned optimizers is particularly relevant for non-stationary environments: as the agent continues learning during its lifetime, appropriate hyperparameter values for an optimizer are likely to change over time and can be impossible to determine in advance (Parker-Holder et al., 2022). Despite this promise, the performance and properties of meta-gradients in the context of reinforcement learning in non-stationary environments have not been systematically studied, which is the focus of our work. The question we focus on is how does the information provided to the meta-optimizer as well as the rate of environment non-stationarity, effect the performance and properties of meta-gradients.

Most of the prior work in this area can be classified within the framework of white- and black-box optimization (see Related Work in Xu et al. (2020) for a more

detailed classification of prior work). Black-box methods express the entire update rule as a rich parametric function, typically a recurrent neural network, and learn the parameters of this function in an end-to-end manner (Xu et al., 2020; Oh et al., 2020; Kirsch et al., 2021). In contrast, white-box methods tune the hyperparameters of the optimization algorithm (Mahmood et al., 2012; Xu et al., 2018; Zahavy et al., 2020). We limit the scope of our analysis to white-box meta-gradient methods for self-tuning hyperparameters, which have to date shown the greatest empirical gains in RL (Xu et al., 2018; Zahavy et al., 2020; Flennerhag et al., 2022).

Since white-box methods are almost completely memory-less and only tune several hyperparameters based on their local influence on agent performance, standard white-box meta-gradients are only capable of tracking good solutions (Sutton et al., 2007) and lack the ability to learn and benefit from past experience. These problems are partly addressed in some recent works in reinforcement (Flennerhag et al., 2020) and supervised (Almeida et al., 2021) learning, which extend the standard white-box formulation by replacing learned hyper-parameters with learned functions of hand-picked *context* features. Broadly speaking, context features can be any low-dimensional statistics of optimization, agent, or environment that carry information about suitable hyperparameter schedules. Examples of such context features in RL are reward histories, temporal difference errors and task beliefs. We refer to this approach as *contextual meta-gradients*. While sharing some similarities with black-box meta-gradients, contextual meta-gradients differ in two important ways: (1) which update functions can be learned is more constrained for contextual meta-gradients (as the update rule can only be changed through hyperparameters) and (2) contextual meta-gradients have access to different kind of information (selected context features instead of entire history of gradients or parameters as is typical for black-box methods).

In Luketina et al. (2022), we hypothesize that: (1) the addition of context to meta-gradients is particularly well-suited for non-stationary environments with repeated structure, as it enables the optimizer to generalize from previously seen contexts and instantly pick good hyper-parameter values; (2) the advantage of all meta-gradients

methods (with or without context) over well-tuned fixed hyperparameter schedules increases with the rate of environment non-stationarity, as different environment conditions may require very different learning hyperparameter values.

To examine the first hypothesis, we compare the performance of meta-gradients with and without contextual information across several environments. Since the addition of contextual information enables the meta-learner to learn hyperparameter schedules that generalize across learning contexts, we look at how increasing the context richness (i.e. amount of information given to the meta-learner) affects the performance.

We find that making the hyperparameters a learned function of context features almost always helps training (Section 6.5.1), although some contexts are much more useful than others (Section 6.5.3). We complement these findings by examining learned schedules and functions of context (Section 6.5.2), where we show that the use of context is necessary for fast adaptation of hyperparameter values in response to the environment changes, and that learned functions of context are meaningful. Lastly, we investigate the adaptation ability of meta-gradients as the rate of environmental non-stationarity increases, with a particular interest in potential advantages of context in highly non-stationary environments (Section 6.5.4). We find that *without* contextual information, meta-gradients provide small to inconsistent advantage over fixed hyperparameter values in highly non-stationary environments. On the other hand, meta-gradients *with* contextual information provide a much more consistent advantage.

6.2 Contextual Meta-Gradients

Contextual meta-gradients (Flennerhag et al., 2022; Almeida et al., 2021)[1] extend white-box meta-gradients by replacing the meta-parameter variables η_k in the update rule with parameterized functions $\eta = g_{\omega_k}(\cdot)$ of context features c_i: $f\Big(\theta_i, g_{\omega_k}(c_i), \mathcal{D}_i\Big)$.

[1]The use of contextual meta-gradients, as studied in this paper, is mentioned in passing in Flennerhag et al. (2022) for one of the experiments, however, it is not the focus of that paper. Contextual meta-gradients are the main contribution and focus of Almeida et al. (2021), although with a different objective and applied in transfer learning setting. See Section 6.4 for a more detailed comparison.

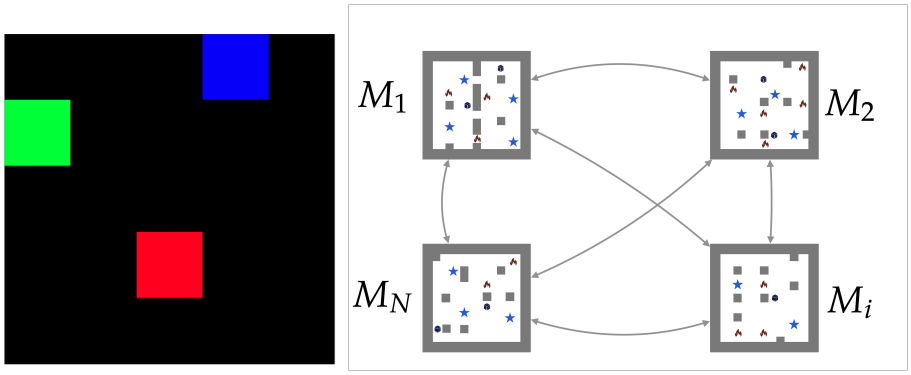
The parameters ω_k of the meta-parameter function $g_\omega(\cdot)$ are learned by optimizing the outer loss, which is now $\mathcal{L}^{(\text{outer})}(\omega, \mathcal{D})$, with the index k referring to the value at outer optimization step k. The meta-parameter function $g_\omega(\cdot)$ at i-th inner update takes context features c_i as inputs, and outputs the value of the meta-parameter η used in the i-th inner update. For example, $g_\omega(\cdot)$ could be a neural network that takes average TD error over i-th batch as input and predicts the coefficient of L_2 regularization used in the i-th inner update. Also note that depending on how the context features are constructed and shared, the changes introduced by contextual meta-gradients allow for different meta-parameter predictions between individual inner updates or even individual samples contributing to the inner loss.

The context features can be any learning or environment statistics that capture information relevant for optimal meta-parameter schedules. Since contextual meta-gradients involve a function approximation in learning the meta-parameter function $g_\omega(\cdot)$, in practice, their usefulness relies on this meta-network being able to generalize with respect to context. For instance, when the learning process exhibits a cyclical pattern throughout the lifetime, contextual meta-gradients can improve upon standard meta-gradients as changes in learning dynamics are predictable and meta-network can learn the association between context and meta-parameter values. For example, if the context indicates there has been a drop in agent's performance, the meta-network $g_\omega(\cdot)$ predicting the rate of exploration can learn to associate this drop with an increase in the rate of exploration.

In our experiments, the context features were chosen to indicate changes in task at hand and the agent performance (e.g. statistics of rewards, TD errors, values), all of which may require different meta-parameter values. Lastly, as in previous work with contextual meta-gradients (Flennerhag et al., 2022; Almeida et al., 2021), the gradients do not propagate through the context features.

6.3 Non-Stationary Environments

To study non-stationarity, we chose to focus on environments with discrete and regular changes in reward and transition function. The regularity of changes



(a) Two Colors. **(b)** Switching MDPs.

Figure 6.1: Illustration of Two Colors and Switching MDPs environments. (a) In Two Colors, the agent (green) is tasked with navigating to either blue or red square. (b) In Switching MDPs, the reward and transition function changes to one of the N predefined randomly generated options at regular intervals.

enables control over the degree of non-stationarity by increasing or decreasing the length of the interval between two changes (Section 6.5.4) and it makes the learned meta-parameter schedules more interpretable (Section 6.5.2). The agent interacts with these environments via a single-stream of experience as in the standard RL formulation (there is only one environment and not multiple copies of it, as is common in distributed RL frameworks). Note that the range of reward in considered environments does not change during a lifetime, hence the context features related to reward (such as those based on reward, value and TD-error) are likely to stay within a certain distribution, which is important for generalization of context functions. The Two Colors environment is visualised in Figure 6.1a and Switching MDPs in 6.1b.

Two Colors. We start with a gridworld environment introduced in Flennerhag et al. (2022), where the agent is tasked with picking up one of the two items. Picking up one of the items results in reward +1 or -1, after which the agent and two objects are randomly re-spawned. Which object carries positive reward changes every 100,000 steps. The agents used in our experiments are memory-less, hence the optimal policy has to be re-learned after each task switch, which requires increasing the rate of exploration after the task switches. The simplicity of this setting allows us to have a clear expectation of what the optimal exploration schedule should

look like, which eases the interpretation of the learned schedules and functions. The dimension of the grid in Two Colors environment is 5×5. The observation space is constructed by concatenating one-hot encodings of x and y coordinates of positions of agents and two other objects. The total dimension of resulting observation space is $3 \times 2 \times 5 = 30$.

Switching MDPs. In our second non-stationary setting, the agent interacts with a sequence of grid worlds, where a new grid world is sampled every 100,000 steps from a set of N grid worlds. The reward function and transition function of each grid world are randomly generated.

We use this environment to study how changes in the environment dynamics changes, in addition to the reward function, affect meta-gradient methods. Additionally, in contrast to the Two Colors environment, two consecutive tasks may not be as different from each other and contain some positive transfer. This implies that in some cases, the optimal exploration strategy upon a task change may not be to explore too much. We experimented with two variants of Switching MDPs with different number of grid-worlds N. In the first case N=4 MDPs so that the agent is repeatedly exposed to the same tasks, and can improve its learning over time. In the second case, N=1000 MDPs, and the agent is very unlikely to experience the same task twice, requiring constantly learning almost from scratch in each MDP. This case is interesting, because we can study the generalization of contextual meta-gradients to unseen contexts. As tasks effectively do not repeat, distribution of context features which are given to the meta-parameter function after each task switch is much more irregular. For example, we don't observe regular periodic drops and increases in mean reward as we do in Two Colors environment.

The reward function for each MDP is generated in the following manner: for each state-action pair there is a 50% chance of zero reward, 20% chance of reward of +1, 20% chance of reward -1, and 10% chance of a random reward sampled uniformly from the interval [-1, 1]. The transition function for each MDP is a standard two dimensional grid world—states are characterized by an (x, y) coordinate, and there

are four actions that move the agent up, left, right, and down unless the intended cell is the edge of the grid or a wall is present. The transition function changes across the N MDPs by the addition of a set of walls randomly placed throughout the grid. Up to 15 walls are placed per-MDP, in sequence, by randomly sampling unoccupied cells (by goal, agent, or another wall).

At the beginning of an experiment, a set of N different MDPs is generated. At regular fixed intervals, the next grid world is sampled from this set uniformly with replacement. The set and the order of samples is uniquely determined by experiment random seed. If N is large compared to the total number of task switches experienced during a lifetime (e.g. $N = 1000$ with a change period $100,000$ and a lifetime of $20M$ steps), the probability of agent experiencing the same task multiple times is small. Note that since we measure the lifetime performance after $20M$ environment steps in our experiments, N does not correspond to the number of different MDPs experienced during this lifetime.

6.4 Related Work

White-Box Metagradients. The focus of our paper is on white-box methods for learning adaptive optimizers (Bengio, 2000; Maclaurin et al., 2015). These methods tune the meta-parameters (i.e. tunable subset of hyper-parameters) of the learning update by computing the gradient of the outer loss with respect to meta-parameters, where the outer loss depends on meta-parameters both directly and through the history of last K parameter updates. In RL, this approach has been used to tune various optimization hyper-parameters, including discount and bootstrap parameters (Xu et al., 2018), off-policy corrections (Zahavy et al., 2020), auxiliary rewards and tasks (Zheng et al., 2018; Veeriah et al., 2021) and weights of rewards in return estimates (Wang et al., 2019).

Contextual Metagradients. Flennerhag et al. (2022) and Almeida et al. (2021) propose learning meta-parameters as functions of context features, with the application in reinforcement and supervised learning respectively. While the focus

of Flennerhag et al. (2022) is on reducing myopia and conditioning problems of meta-gradients by developing an improved outer loss, in their experiments on non-stationary environments, they parameterize meta-parameters as a function of contextual features (in their case, reward statistics). However, this is only mentioned in passing, and the authors do not study the importance of including contextual information or provide a comparison to alternative contextual features. In Almeida et al. (2021), learning meta-parameters as a function of features is the primary focus. In contrast to most prior work in white-box meta-gradients, Almeida et al. (2021) formulate optimization of meta-parameters as a reinforcement learning problem. The meta-parameter function is a policy modifying the current values of corresponding meta-parameters, trained on distribution of source tasks with a designed reward function. The context features are selected to facilitate transfer of learned optimizers to the target tasks. In contrast to our work, their focus is on achieving high performance on supervised learning tasks instead of analysis of meta-gradients in non-stationary RL environments.

Black-Box Metagradients. Alternatively, black-box methods parametrize the entire update rule as a neural network, and learn it from scratch by training on distribution of supervised (Andrychowicz et al., 2016) or reinforcement learning (Kirsch et al., 2020; Oh et al., 2020; Kirsch et al., 2021) tasks. In RL, most of these methods require training over multiple tasks or lifetimes, which makes them not directly applicable to our setting of interest. The most relevant work is Xu et al. (2020), which develops a black-box method that trains an optimizer over a single lifetime. In black-box methods, the inputs to the learned optimizer functions are typically a history of parameters and gradients, but more similar to context features explored in our work, they can also include the entire rollout trajectories (Xu et al., 2020; Oh et al., 2020). However, these works do not explore the importance of selecting inputs for the learned optimizer and its effects on training in non-stationary environments.

Contextual MDPs. Another related line of work is that of Contextual MDPs (Hallak et al., 2015; Modi et al., 2018; Belogolovsky et al., 2021). In this setting, contexts, which define the reward and transition kernel, are sampled from a distribution. This formulation allows the study generalisation, e.g., we can learn a mapping from contexts to MDPs and ask how well should this mapping generalize to unseen contexts; see (Kirk et al., 2021) for a recent survey on the topic. Our work, on the other hand, does not make such an assumption about the world and instead focuses on a single, but non-stationary MDP. It might be possible to think about some specific non-stationary MDPs as special cases of contextual MDPs, but we leave this exciting direction for future work.

6.5 Experiments

We designed the experiments with the goal of answering the following questions about the behaviour of meta-gradients in non-stationary environments:

Do meta-gradients benefit from contextual information? We hypothesise that ability to learn meta-parameter schedules as functions of contextual information will enable faster adaptation in non-stationary environments, as the optimizer can leverage knowledge from previously seen contexts and instantly utilize good meta-parameter values without needing to slowly tune them. We compare the performance of contextual meta-gradients and baselines in Section 6.5.1.

What functions of context are learned in this process? To explain the performance difference between methods, we examine the meta-parameter schedules during training and the learned meta-parameter functions of context in Section 6.5.2.

What should the contextual information be? The choice of context features could be essential for learning schedules that generalize over the training. Here we look into increasingly rich contexts: what is the effect of adding particular candidate contexts and can too much information be detrimental for generalization? We shed light on these questions in Section 6.5.3.

How does the performance of meta-gradients depend on the rate of non-stationarity? As the rate of change of the environment increases, we hypothesized that the ability to adapt the meta-parameters becomes more important. Furthermore, the addition of context to meta-gradients should lead to further advantages, as the they enable faster adaptation. The advantage of meta-gradients under different rates of non-stationarity is examined in Section 6.5.4.

Experimental Axis

To ensure the conclusions we make are robust, we run the experiments along several axis: (i) agents in inner loop, (ii) outer loop objectives, (iii) context features:

Agents. We use two different kinds of RL agents: Actor-Critic (AC) (Sutton et al., 1999b) and $Q(\lambda)$ (Peng & Williams, 1994). The parameters of an AC agent are updated every 16 environment steps, and those of $Q(\lambda)$ agent at each step. If using AC agent, we tune the coefficient of entropy loss, whereas if using $Q(\lambda)$ agent, we tune the ϵ parameter of ϵ-greedy exploration. The details of the agent architectures can be found in Appendix C.3.

Outer Loop Objectives. In experiments combining meta-gradients (MG) with AC agents, the outer loss is a sum of policy loss $\mathcal{L}_\pi$ and a target entropy loss $\mathcal{L}_{\text{ent}}$ (the weight of this entropy loss is a fixed hyperparameter $\alpha_{\text{ent}}^{(outer)}$):

$$\mathcal{L}_{\text{MG}}^{(\text{outer})} = \mathcal{L}_\pi + \alpha_{\text{ent}}^{(\text{outer})} \mathcal{L}_{\text{ent}}. \tag{6.1}$$

If using Bootstrapped Meta-Gradients (BMG) (Flennerhag et al., 2022), the outer loss is KL divergence between the target policy $\pi_{\hat{\theta}}$ and K-step bootstrap π_{θ_K}:

$$\mathcal{L}_{\text{BMG}}^{(\text{outer})} = KL(\pi_{\hat{\theta}} || \pi_{\theta_K}), \tag{6.2}$$

where the target policy is the policy reached after $K + L - 1$ steps of inner loop optimization. For a longer description of BMG objective see Section 2.4.1 and for a description of how to make BMG objective differentiable with respect to ϵ when used with $Q(\lambda)$ agents, see Appendix C.1.

Feature Type	$f_i^{(feat)}$	Feature Dimension
Reward	r_t (mean & std)	$2 \times H$
Value	$v(s_t;\theta_i)$ (mean & std)	$2 \times H$
TD Error	$r_t + \gamma v(s_{t+1};\theta_i) - v(s_t;\theta_i)$ (mean & std)	$2 \times H$
Action Probabilities	$\pi(a\|s_t;\theta_i)$ (mean & std)	$8 \times H$
States	s_t (mean & std)	$60 \times H$
Gradient Cos Distance	$1 - \frac{\nabla_\theta \mathcal{L}_{i-1} \nabla_\theta \mathcal{L}_{i-2}}{\|\nabla_\theta \mathcal{L}_{i-1}\| \|\nabla_\theta \mathcal{L}_{i-2}\|}$	H
Meta-Parameters	η_{i-1}	H

Table 6.1: Summary of all context features used in experiments with AC agents. When description of $f_i^{(feat)}$ includes mean & std, we are indicating that $f_i^{(feat)}$ is mean and standard deviation of the specified quantity, over the rollout data used to compute i-th update. Here we denoted the gradient with respect to inner loss at i-th update with $\nabla_\theta \mathcal{L}_i$.

Feature Type	$f_i^{(feat)}$	Feature Dimension
Reward	r_t	H
Value	$q(a_t, s_t;\theta_i)$	H
TD Error	$r_t + \gamma \max_a q(s_{t+1}, a;\theta_i) - q(s_t, a_t;\theta_i)$	H

Table 6.2: Summary of all context features used in experiments with $Q(\lambda)$ agents.

Context Features. We compare two kinds of contextual features, simple features referred to as *Reward* (only using reward statistics) and more complex features referred to as *Rich* (using reward, TD error and value statistics).

If the context is *Reward*, each context feature is either a history of the last H observed rewards ($Q(\lambda)$ agent) or a history of mean rewards observed in each of the last H rollouts (AC agent). If the context is *Rich*, we compute the same history for reward, TD error and values. In AC experiments, rich context also includes a history of last H standard deviations in addition to means. We use $H = 10$ in experiments with AC agents, and $H = 100$ in experiments with $Q(\lambda)$ agents. Each context feature is normalized and scaled to be in the [-1, 1] range, before being concatenated and passed down to the learned meta-parameter function.

More specifically, the context features are constructed in the following way. At each update step i, the meta-parameters are computed by feeding context features c_i into a meta-parameter function. The context features are a concatenation of one or several different types of features (e.g. reward, value, TD error), the type of

features used in each experiments is described in the corresponding experiment's section. For example, when the context is specified as *Reward*, the input to the meta-parameter function is $c_i = \left[c_i^{(reward)}\right]$, and when the context is *Rich*, the input to the meta-parameter function is $c_i = \left[c_i^{(reward)}, c_i^{(value)}, c_i^{(td-error)}\right]$. For each different feature type (here denoted with "feat"), the features are a history of size H:

$$c_i^{(feat)} = \left[\hat{f}_i^{(feat)}, \hat{f}_{i-1}^{(feat)}, ..., \hat{f}_{i-H+1}^{(feat)}\right], \tag{6.3}$$

where $\hat{f}_i^{(feat)}$ is a quantity computed using statistics at i-th update step. Each of these quantities has been normalized (we used the statistics observed over a lifetime, but more generally, one could use a running average as an estimate) and passed through a $tanh(\cdot)$ function to ensure all features are in the [-1, 1] range. We will refer with $f_i^{(feat)}$ to quantities computed before this transformation. For example, in experiments described with AC-BMG-Reward, each $f_i^{(reward)}$ is a mean of rewards r_t from the rollout D_i which was used to compute i-th update. In experiments described with $Q(\lambda)$-BMG-Reward, because i-th update for $Q(\lambda)$ agents is calculated using data from only one environment step, $f_i^{(reward)}$ is the reward r_t from only that step.

The summary of all different feature types used in experiments with AC agents, including how $f_i^{(feat)}$ is defined for that feature type and dimensions of that feature, can be be found in Table 6.1. Note though that when the feature is specified as *Reward* (AC-MG-Reward and AC-BMG-Reward), we only used mean of rewards, and the corresponding feature dimension is H. The summary of all different feature types used in experiments with $Q(\lambda)$ agents can be found in Table 6.2.

Hyperparameters. In each experiment, **we report only the results for the best hyper-parameter setting**. For the baselines (which do not adapt the meta-parameters), we include the meta-parameter of interest into the hyperparameter sweep. For MG objectives, we sweep over the meta learning rate, meta rollout length (K) and target entropy loss coefficient ($\alpha_{\text{ent}}^{(\text{outer})}$). For BMG methods, we sweep over meta learning rate, meta rollout length (K) and target rollout length (L). For

	Context Features		
Method	None	Reward	Rich
AC	1.24 (0.09)	N/A	N/A
AC-MG	1.29 (0.08)	1.58 (0.06)	1.62 (0.10)
AC-BMG	1.32 (0.08)	1.74 (0.03)	**1.79 (0.07)**

Table 6.3: Two Colors with AC Agent. We compare meta-gradient agents with MG and BMG outer-loop objectives (AC-MG, AC-BMG), against a baseline with fixed meta-parameters (AC). For each meta-gradient method, we report results without the use of context features (*None*), with only reward features (*Reward*), and with rich context features ((*Rich*). Mean and standard deviation of total return after 10M steps (10 seeds). Only meta-gradient methods *with* context features significantly outperform the AC baseline.

	Context Features	
Method	None	Reward
Q(λ)	0.58(0.07)	N/A
Q(λ)-BMG	1.78(0.03)	**1.93(0.03)**

Table 6.4: Two Colors with $Q(\lambda)$ Agent. We compare meta-gradient agent with BMG outer-loop objective ($Q(\lambda)$- BMG) against a baseline with fixed meta-parameters ($Q(\lambda)$). For the meta-gradient agent, we report results without the use of context features (*None*) and with only reward features (*Reward*). Mean and standard deviation of total return after 10M steps (10 seeds). Both meta-gradient methods significantly outperform the $Q(\lambda)$ baseline, with the highest performance obtained by meta-gradients with Reward context features.

fairness, the size of hyperparameter sweeps is the same for all meta-gradient methods. For more details on the hyper-parameter sweeps and values, see Appendix C.3.

6.5.1 Do Meta-Gradients Benefit from Adding Contextual Information?

We start the experiments by testing how adding contextual information to meta-gradients effects training. If our hypothesis is true, we expect to see the advantages of adding contextual information across different agents and outer loop objectives. Results in bold indicate the best mean performance among the compared methods.

Two Colors: Actor-Critic Agent

We first look at the AC agents trained on Two Colors (described in Section 6.3). In Table 6.3, we report the mean and standard deviation of the total reward

	Number of MDPs	
Method	4	1000
$Q(\lambda)$	14.77 (1.78)	12.03 (0.96)
$Q(\lambda)$-BMG	14.79 (1.56)	11.71 (0.71)
$Q(\lambda)$-BMG-Reward	15.00 (2.21)	13.36 (0.68)
$Q(\lambda)$-BMG-Rich	**16.03 (1.40)**	**13.63 (0.64)**

Table 6.5: Switching MDPs with $Q(\lambda)$ Agent. Mean and standard deviation of total return after 20M steps (10 seeds). Contextual information was necessary to gain a significant improvement over the baseline with meta-gradients. The improvement is greater when the number of different MDPs (i.e. variety of tasks experienced during a lifetime) is greater.

after 10M environment steps. The history length H for context features is 10 in these experiments.

We find that all meta-gradient methods outperform the baseline with fixed meta-parameters (AC) and consistent with reports in Flennerhag et al. (2022), BMG performs better across methods. More importantly, the addition of context (see columns with Context Features set to Reward or Rich) is crucial for obtaining significant improvement over the baseline, with the richer features performing slightly better.

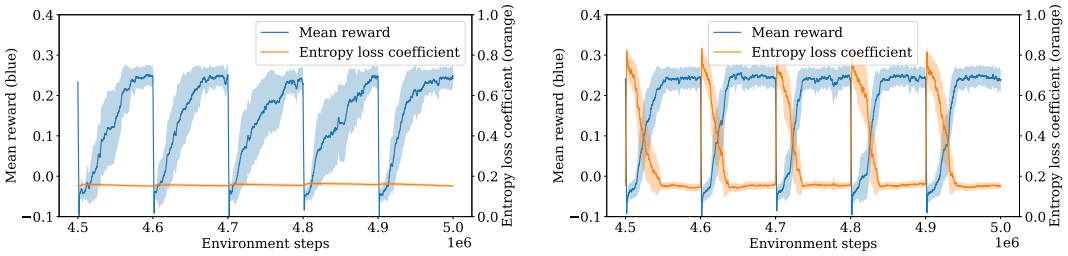
Two Colors: $Q(\lambda)$ Agent

Next, we look at the $Q(\lambda)$ agents on Two Colors. In Table 6.4, we again report the mean and standard deviation of total returns after 10M environment steps. We tune the ϵ parameter of ϵ-greedy exploration with only BMG objective, as the updated value function in this case is not a differentiable function of ϵ and consequently can not be straightforwardly optimized with regular MG objectives. The history length H for context features is 100 in these experiments.

We find that meta-gradients significantly outperform the non-adaptive baseline and the addition of context further boosts the performance.

Switching MDP: $Q(\lambda)$ Agent

Next, we look at $Q(\lambda)$ agent trained on Switching MDPs. In Table 6.5, we report the mean and standard deviation of total rewards after 20M environment steps.



(a) BMG. **(b)** BMG-Reward.

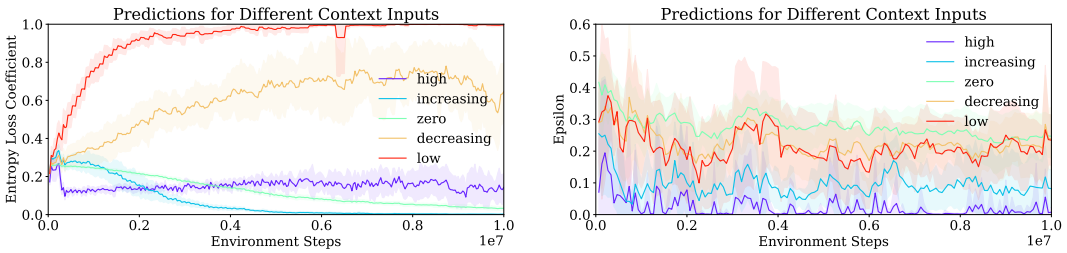
Figure 6.2: Entropy coefficient (orange) vs. mean reward (blue) during training for: (a) AC with BMG, (b) AC with BMG and reward context. We report values at each timestep in the middle of training averaged over 10 seeds. The error margins represent 95% confidence intervals. The drops in mean reward (blue) at regular correspond follow the task switches. Without access to contextual information, meta-gradients learn an almost constant entropy schedule. By adding reward context, learned entropy schedule strongly responds to drops in mean reward.

We report only the experiments with $Q(\lambda)$ agents since AC agents were performing very poorly on this environment. Again, we tune only the ϵ parameter. For these experiments, we also vary the number of different MDPs available in the environment (see Section 6.3 on the meaning and relevance of these environment variations).

We compare the results under the two environment variants of interest (4 and 1000 MDPs) in Table 6.5. The addition of context was necessary to obtain significant improvements with meta-gradients, with Rich context resulting in the biggest improvement. The improvement is also much more significant in the regime where the MDPs effectively do not repeat (1000 MDPs). Because the consecutive tasks will typically be more similar compared to Two Colors, we hypothesize there is less need to re-learn in the regime with a small number of tasks (4 MDPs), requiring less adaptation of meta-parameters.

6.5.2 What Meta-Parameter Schedules and Functions are Learned?

In this section, we wish to examine the predictions of the learned meta-parameter functions throughout training. For the ease of analysis, we focus on BMG with Reward context features (BMG-Reward) while only tuning one meta-parameter.



(a) Two Colors, AC-BMG-Reward. **(b)** Switching MDPs, $Q(\lambda)$-BMG-Reward

Figure 6.3: Predicted values of exploration meta-parameters for five qualitatively different reward context features as inputs to the meta-parameter function, as measured during training in: (a) Two Colors with AC-BMG-Reward, (b) Switching MDPs (1000 MDPs) with $Q(\lambda)$-BMG-Reward. Each curve is averaged over 10 random seeds with the error margins representing one standard deviation. In (a), the α_{ent} is predicted when the rewards are the lowest, and the lowest when the rewards are increasing. In (b), the highest ϵ is predicted for lowest or zero rewards, and the lowest for highest rewards.

Learned Schedules

We look at the relationship between the learned meta-parameter schedule and the observed rewards for AC agents with BMG trained on Two Colors. In Figure 6.2a, meta-gradients do not rely on context features, and in Figure 6.2b, meta-gradients utilize Reward context features. The two curves in both figures are: predicted entropy loss coefficient during training (orange curve) and mean reward over rollout (blue curve).

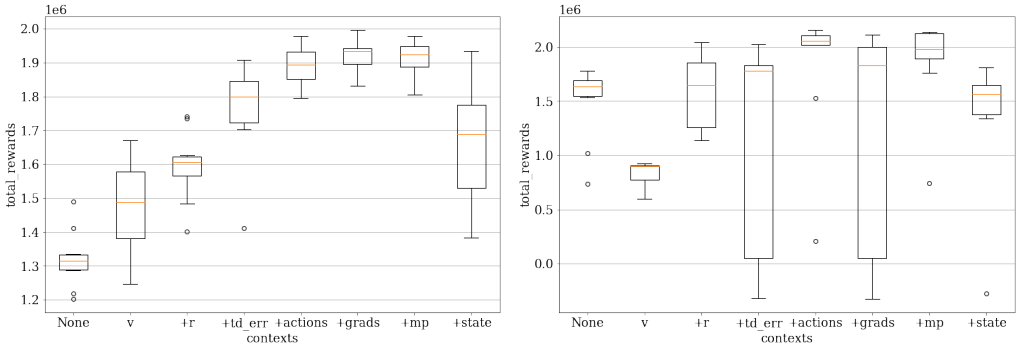
When meta-gradients do not have access to reward context (Figure 6.2a), the mean reward takes longer to recover after the drop following each task switch. Furthermore, entropy coefficient is almost constant. Note that this can not be explained by too low meta learning rate: the results shown here are under the best hyper-parameter configuration and the best performing meta learning rates were never the highest values in the sweep. In contrast, when the reward context is added (Figure 6.2b), the entropy coefficient rapidly increases after each task switch to allow the agent to explore. As a result, the mean rewards of the agent are better as we observed in the previous section, suggesting that the addition of context is crucial for obtaining fast adaptation.

Learned Functions

Next, we look at how the context to meta-parameter mapping itself changes during a lifetime. Our methodology is inspired by visualization of model trajectories in Erhan et al. (2010). Due to low dimensionality of inputs and outputs of meta-parameter function, we can select a small number of representative context inputs and for each, track the corresponding meta-parameter prediction during training. Note that we do not train on these context inputs, we just log the output for each while the meta-parameter function is trained as usual.

The five context inputs were selected to represent qualitatively different learning situations: constantly high context values (*"high"*), monotonically increasing from lowest to highest value (*"increasing"*), constant zero (*"zero"*), monotonically decreasing between the highest and lowest value (*"decreasing"*) and constantly low context (*"low"*). For example, for rewards bounded in $[-1, 1]$ and $H = 3$, the context inputs are: "high" $= [1, 1, 1]$, "increasing" $= [-1, 0, 1]$, "zero" $= [0, 0, 0]$, "decreasing" $= [1, 0, -1]$ and "low" $= [-1, -1, -1]$. In our experiment, $H = 10$ and the range is $[-1, 1]$ due to pre-processing of context features. Note that some of the context inputs have the same mean (e.g., *"increasing"*, *"zero"* and *"decreasing"* all have mean zero), however they capture qualitatively different behaviors. For example, the *"decreasing"* input corresponds to a context likely observed just after switching the task, while the *"increasing"* probe is likely observed as the agent starts improving in a new task.

We visualize the learned meta-functions of AC agent trained with BMG-Reward on Two Colors in Figure 6.3a, and those of $Q(\lambda)$ agent trained with BMG-Reward on Switching MDPs (1000 MDPs variant) in Figure 6.3b. First, we can observe that different inputs results in very different meta-parameter predictions and that the meta-parameter functions seem to converge during training (the small local changes are likely due to non-stationary training distribution), hence addition of context enables convergence instead of just tracking as for vanilla MG. Next, we look at the values of predicted meta-parameters for each of the five context inputs. We can observe that the differences in predictions are sensible: in Figure 6.3a, the



(a) AC-BMG (α_{ent}) **(b)** AC-BMG (α_{ent}, α_{L2})

Figure 6.4: Performance of meta-gradients as a function of increased context richness: (a) AC-BMG with tuned entropy loss coefficient, (b) AC-BMG with tuned entropy and L_2 loss coefficients. Total reward at 10M environment steps (10 seeds). Starting without contextual information (None), in each column, from left to right, we add the statistics of the following quantities as features to the context: value (v), reward (+r), TD error (+td_err), action probabilities (+actions), cosine distance between gradients (+grads), previous values of meta-parameters (+mp) and state visitation (+state). With some exceptions, adding more information leads to increase in mean performance.

entropy loss coefficient is highest when the rewards are at low or decreasing, and the lowest when the rewards are high or decreasing; whereas in Figure 6.3b, the exploration is highest when the rewards are low, decreasing or zero and lowest when the rewards are high. The predictions are different for context inputs with the same mean value ("increasing", "decreasing", "zero"), indicating learned function responds to more complex patterns than just mean values of features. Lastly we note that for Switching MDPs, we see more variability in learned functions across different training seeds. This is likely due to a more complex interaction between encountered tasks and stochasticity in generating and sampling tasks (i.e. how much transfer there is between consecutive tasks will depend on which two tasks are sampled).

6.5.3 How Important is the Choice of Context?

In the following set of experiments, we inspect how making the context more rich effects performance. For this goal, we focus on Two Colors with the AC agent and BMG objective. We consider two variants; one that tunes the entropy coefficient (Figure 6.4a) and one that tunes the coefficients of both the entropy and the L_2 loss (Figure 6.4b).

The x-axis of each figure corresponds to the degree of richness of the context. On the left, we start with no context, and then add one-by-one the statistics of following quantities as context features: value, reward, TD error, action probabilities, cosine distance between last two gradients, past meta-parameter predictions and state visitation statistics. For each of these quantities (except cosine distance and past meta-parameter predictions), the features are a history of means and standard deviations calculated over the last H rollouts. For cosine distance and past meta-parameter predictions, the features are just a history of their last H values. Since the number of features is large, we decrease the history length H from 10 to 4. If we were to use $H = 10$, the dimension of richest context input would be 660, which could make the optimization of meta-function too challenging.

The only cases where context features harm the performance are inclusion of statue visitation features in Figures 6.4a and 6.4b, and relying on just value features in Figure 6.4b. In the case of adding state visitation features, the performance drop is likely due to over-fitting. State features have much higher dimensionality compared to other features, yet they are not informative of the agent performance and hence good meta-parameter values. We did not find evidence of over-fitting for other features. Generally speaking, richer features led to better performance as meta-parameter function has more signal to leverage and can learn to rely on features that carry information about good meta-parameter values while ignoring the others. Note that the features that seem to help the most (reward, TD error), are strongly correlated with agent performance.

6.5.4 How do Meta-Gradients Perform under Different Rates of Non-Stationarity?

Lastly we look at how the degree of non-stationarity in the environment effects the performance of meta-gradients. We control the degree of non-stationarity by changing how often the tasks switch – shorter change periods correspond to higher rates of non-stationarity. In all of the Figures included in this section, the medians and interquartile ranges are computed over 10 random seeds.

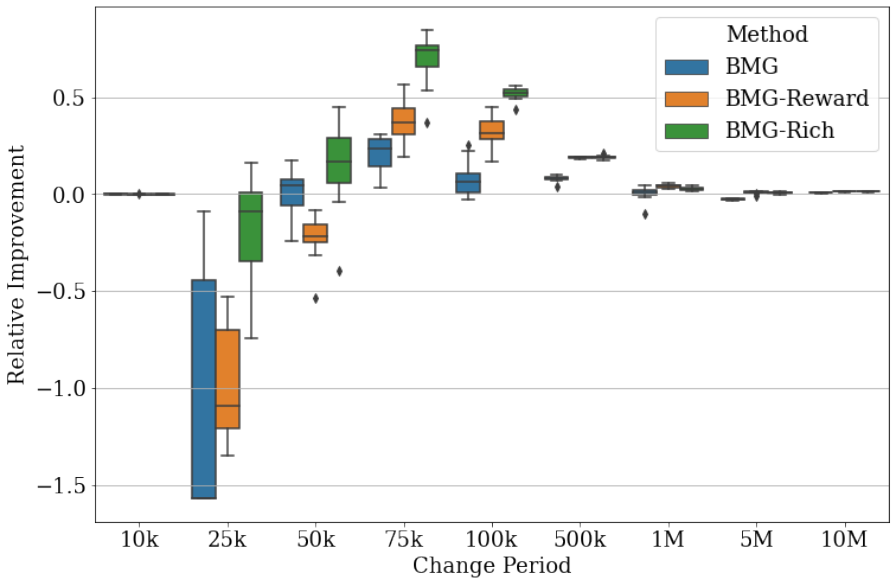
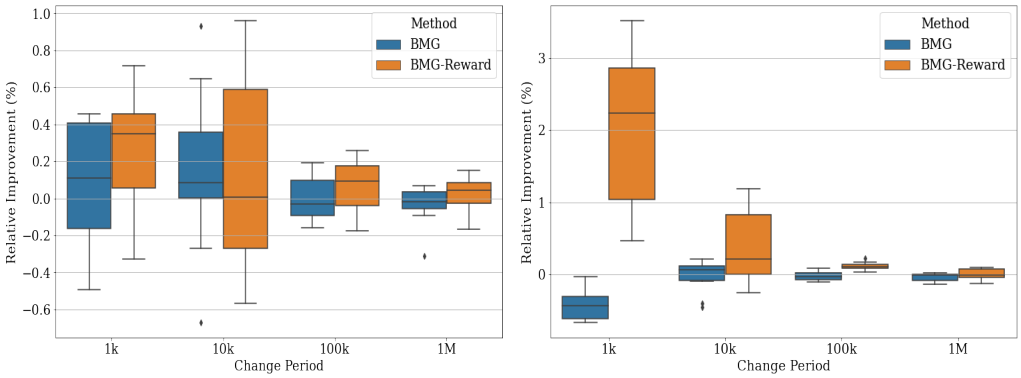


Figure 6.5: Two Colors: Relative improvement over the fixed meta-parameter AC baseline and under different rates of non-stationarity. The regime in which meta-gradients provide a significant advantage over the baseline is the greatest for meta-gradients with rich context [50k, 500k]. All meta-gradient fail to beat the baseline when the rate of non-stationarity is too high (25k).

Two Colors

In this experiment, we compare the relative improvement of different meta-gradient methods over the AC baseline for various non-stationarity rates (Figure 6.5). The relative improvement is defined as percent increase or decrease in performance (as measured in total reward after 10M steps) over the mean performance of the baseline (with the same non-stationarity rate). The absolute (non-relative) values of all the methods can be found in the Appendix C.4.2. We use BMG as an outer loop objective and tune entropy rate coefficient. The compared methods are: BMG (no context), BMG-Reward and BMG-Rich. As before, we report only the results for the best hyper-parameter configurations.

We find that when the environment changes too rapidly, at around 50,000 steps, the performance of meta-gradients deteriorates. We expected to find that improvement brought on by meta-gradients is greater when the environment changes more rapidly. This expectation was met up to a point: BMG with a rich context were less affected in addition to outperforming context-less meta-gradients for any



(a) 4 MDPs **(b)** 1000 MDPs

Figure 6.6: Relative improvement with meta-gradients over the $Q(\lambda)$ baseline under different rates of environment non-stationarity: (a) for Switching MDPs with 4 MDPs, (b) Switching MDPs with 1000 MDPs. Meta-gradients with Reward context provide a more reliable performance boost, with the biggest improvement when the rate of non-stationarity and the number of different tasks are high.

rate of change, indicating that providing more information can provide enough signal to enable meta-learning even in this regime.

Switching MDPs

Figure 6.6a and Figure 6.6b present the relative improvement of meta-gradients over the $Q(\lambda)$ baseline in the Switching MDPs environment (for $N = 4$ and $N = 1000$ respectively). The absolute (non-relative) values can be found in the Appendix C.4.2.

We find that the use of context becomes more useful as we increase the rate of environment non-stationarity and the number of different tasks, where fast adaptation of meta-parameters becomes more relevant.

6.6 Discussion

While we observe significant improvements through the use of contextual meta-gradients in the environments we tested, we also observe an increase in stability issues when trying to tune multiple hyper-parameters. For the ease of analysis, in most of our experiments, we focus on tuning the rate of exploration, however we also experimented with tuning weight decay in addition to rate of exploration. While on average we observe an improvement in both cases, there is a significantly higher

percentage of runs that collapse during training when tuning two hyperparameters (see the increase in variability of results in Figure 6.4b: while most runs outcompete the non-contextual meta-gradients, some runs also collapse to close to zero or negative total lifetime rewards). This observation was even more pronounced in our initial experiments on tuning even more hyperparameters (such as learning rate and discount), where the majority of runs collapsed. In contrast, the non-contextual meta-gradients do not appear to suffer from the same problem (Zahavy et al., 2020), neither do contextual meta-gradients trained with different objectives and in multi-task settings (Almeida et al., 2021). At the same time, instability is a well-known issue in training black-box optimizers (Metz et al., 2019, 2022; Harrison et al., 2022). Investigating the sources of instability in different methods of learning optimizers is an interesting and important future work, however it is beyond the scope of our work.

Another important assumption made in our work is that the range of rewards as well as values of other context features does not change during the lifetime (see Section 6.3). This might be crucial for the effectiveness of contextual meta-gradients in non-stationary settings, as it means the learned meta-parameter functions will not encounter significant distributional shifts. Moreover, abrupt changes in the underlying environment dynamics warrant re-learning of policies and values, meaning similar optimizational circumstances will repeat over the lifetime.

We ran some initial experiments on a small selection of Atari environment (Mnih et al., 2013) (see Appendix C.4.3), where we modify the codebase of Zahavy et al. (2020) and Flennerhag et al. (2022) to support contextual meta-gradients. While under sufficiently low meta-learning rate we do not encounter previously mentioned stability issues, we also do not significantly exceed the performance of non-contextual meta-gradients when averaged over tasks. The hyper-parameter schedules for Atari (see Appendix C.4.3) do not demonstrate the type of regularity observed in Figure 6.2b. This is because in Atari, the underlying environment does not change, the policies and values do not need to be re-learned. Hence the context features observed early in training will not be seen again later in training, diminishing the usefulness of contextual meta-gradients. Moreover, the abrupt and regular environmental

changes characterising environments used in this work (while useful for our analysis) may not be encountered in many practical applications involving environment non-stationarity. Hence future work will be needed to examine the performance of contextual meta-gradients in non-stationary environments with smoother changes.

6.7 Conclusions

We studied the performance and properties of white-box meta-gradients in non-stationary environments. To study the effect of adding contextual information to the learned optimizer, we focused on formulations of meta-gradients where the learned meta-parameter values are functions of selected context features. We found that adding more contextual information is almost always beneficial for lifetime performance. Inspection of learned meta-parameter schedules and functions provides evidence of faster adaptation for meta-gradients with contextual information and convergence of meta-parameter functions over training. When looking at the effect of increasing the rate of non-stationarity, we find that the meta-gradients without context, in contrast to meta-gradients with context, do not offer a large consistent advantage over fixed meta-parameter schedules. An interesting avenue for future research are studies of contextual meta-gradients in continual supervised learning setting and in non-stationary environments with less repeatability of context features.

7
Afterword

This thesis makes several contributions towards more effective transfer learning in sequential decision-making problems. Across four distinct works, we have proposed and analyzed strategies for incorporating natural language data, designing architectures for compositional generalization, utilizing modest supervision to shape unsupervised skill discovery, and enabling optimization algorithms to rapidly adapt in non-stationary environments.

In Chapter 3, we categorized and reviewed various ways in which language and language data can be integrated with sequential decision making, while also revealing gaps in the literature and charting promising future directions. In particular, we advocated for the greater use of LLMs and development of semantically complex research environments with real-world language. Recent works that develop methods for improving exploration and planning through the use of LLMs lend credence to these suggestions. Such lines of work are becoming increasingly relevant as the latest generations of language models are starting to exhibit reasoning and planning capabilities, as well as demonstrate knowledge of various professional and academic domains, with the implications for their usefulness in guiding agents. Our case for the development of standardized benchmarks to better evaluate such approaches continues to have merit.

Our work in multi-modal and multi-task settings (Chapter 4) introduces a modular architecture aimed at facilitating compositional generalization across combinations of different types of observations, actions, and instructions. Controlled experiments demonstrate strong zero-shot transfer on held-out compositions, suggesting that modular design is an effective choice for multi-modal and generalist agents. Results with particularly challenging held-out combinations indicate potential of modular designs for sim-to-real transfer, while the results on new observations spaces demonstrate advantages in lifelong learning systems. Nevertheless, the simplicity of the environment we developed for this work——while enabling a focused and controlled experimentation—does not capture many challenges encountered in real-world applications. Real-world problems may involve a larger divergence and diversity of tasks (one may aim to develop a single agent capable of solving and transferring across much wider sets of tasks, encompasing, e.g., both NLP and robotics tasks), high levels of environment noise, partial observability, uneven dataset sizes, and more. While our work primarily focused on drawing clear conclusions through controlled experimentation, future work should examine the performance of such architectures on larger scale environments more representative of real-world challenges.

For unsupervised skill discovery (Chapter 5), we introduce an approach that transfers knowledge from a small and select set of source tasks to focus exploration on state dimensions most predictive of returns. Experiments show this method yields skills that are diverse in functionally meaningful ways, unlike standard unsupervised objectives. The technique illustrates how modest supervision can help shape representations to steer unsupervised learning towards more useful behaviors. Directions for the future work include improving on the several weaknesses of our approach, including better ways of ensuring sufficient state space coverage during pre-training (e.g., via the use of exploration bonuses), as well as adapting the approach to non-stationary environments by iterating between phases of skill learning and skill usage. While our work explores the use of value function features to determine diversity of skills, another interesting direction of future work involves

investigation and comparison of alternative types of features, such as those derived from large vision-language models, to determine diversity for agents with visually complex observations.

Finally, in studying meta-gradients for non-stationary environments (Chapter 6), we find that conditioning the hyperparameter adaptation on contextual features significantly aids the agents' adaptation throughout the lifetime. We emphasize the strengths of meta-gradients when paired with rich contextual cues. The detailed analysis we provided suggests features based on rewards and TD errors as particularly good candidates for the context features. As in Chapter 4, for the purposes of clear analysis, our work is done on relatively simple and controllable environments, where the non-stationarity is regular and the changes experienced over a lifetime are limited. Hence an important avenue for future work is exploring the performance and limitations of contextual meta-gradients in settings with less regularity and larger changes over a lifetime, that challenge the contextual meta-gradients with out-of-distribution variations.

Together, the work presented in this thesis showcases the versatility of transfer learning in tackling significant challenges in sequential decision-making. It also demonstrates how the strategic use of language, inductive biases, modest supervision, and learned adaptation can expand the reach of reinforcement learning methods to new tasks and domains. While many open questions remain, the techniques and analysis put forth in this thesis contribute building blocks towards the vision of generalizable, continually learning agents that leverage large multi-modal datasets.

Appendices

A

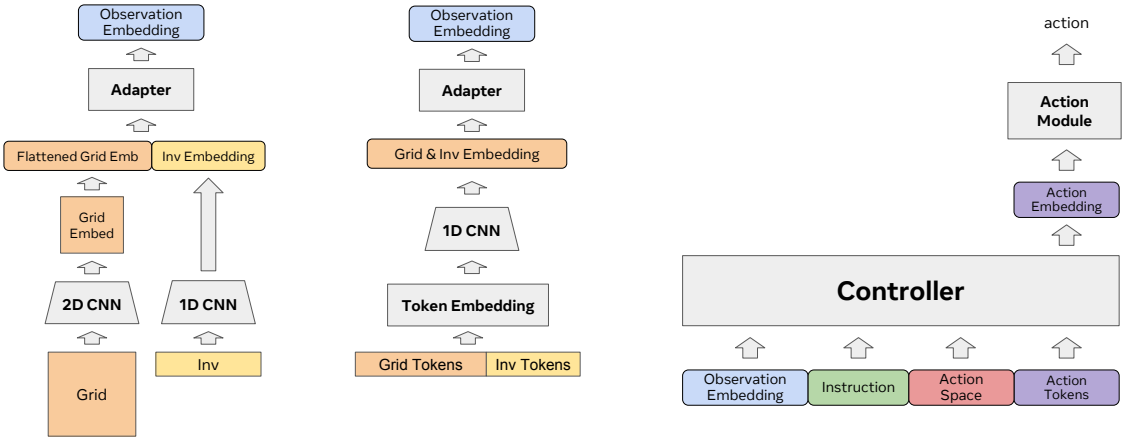
Appendix: Compositional Interfaces for Compositional Generalization

Contents

A.1 Environment

In this section, we provide additional details about the construction and dynamics of the environment. When initiating the environment, we first sample the initial state: the number of objects is sampled uniformly from $[1, 4]$, positions of the objects and the agent are sampled uniformly without replacement. The properties of objects (color, shape) are also sampled uniformly without replacement. The instruction is sampled next, with the constraint that sampled instruction can be completed given the initial state.

We constrain the environment such that two objects can not be in the same



(a) Perception Module (Grid Observations) **(b)** Perception Module (Token Observations) **(c)** Controller and Action Module

Figure A.1: COIN Modular Architecture.

position (e.g. using the Drop action on top of another object will have no effect), however, the agent can be in the same position as an object. In the Grid observation space, if the agent is in the same position (x, y) as an object, we sum the two one-hot representations at (x, y). The color of the agent is always grey.

The inventory has size 4, and the objects are added or removed from the inventory in the last-in, first-out order. Using the Done action in any state other than the goal state will have no effect. Using a movement action that would lead the agent outside the grid also has no effect.

A.2 Architecture

An illustration of the architecture can be seen in Figure A.1. For non-token observation spaces, the perception module is parametrized as in Figure A.1 (a). For observation space Grid, the 2D convolutional network consists of layers of 2D convolutions followed by ReLu non-linearity. The dimensions of output channels in the final convolution correspond to the dimensions of the token embedding of the controller. For the observation space List, the 2D convolution in Figure A.1 (a) is replaced by 1D convolution (same architecture as for inventory embedding: two layers of 1D convolution followed by ReLu non-linearity; with the dimensions of the output channels corresponding to the dimensions of the token embedding of the

controller, which is 768), whereas for the image observation spaces Top View and Side View, the 2D convolution is replaced by a pre-trained ResNet-18 [1] network. For token observation spaces Text and Symbol, the grid and inventory observations are embedded together (see Figure A.1 (b)): each token is first represented using the pre-trained token and positional embeddings, then processed using three layers of 1D convolutions followed by ReLu non-linearity (with kernel sizes 7, 2 and 1 respectively). The resulting output is then passed through a layernorm. We found this architecture to work the best with token observational spaces.

For each observation space O, the resulting inventory and grid embeddings are flattened and concatenated to a sequence $\mathbf{v}$ of length $L^{(O)}$ and passed through an adapter module, resulting in a sequence $\mathbf{h}$ of length L. The adapter module is a simple attention layer of the form:

$$\mathbf{h}_j = \sum_i^{L^{(0)}} \alpha_{i,j}^{(O)} (\mathbf{v}_i + \mathbf{b}_i^{(O)}) \tag{A.1}$$

$$\alpha_{i,j}^{(O)} = softmax(\frac{\mathbf{w}_j^{(O),T} \mathbf{v}_i}{\sum_{i'}^{L^{(0)}} \mathbf{w}_j^{(O),T} \mathbf{v}_{i'}}), \tag{A.2}$$

where $\mathbf{h}_j$ is the j-th element of list $\mathbf{h}$, $\mathbf{v}_i$ is i-th element of list $\mathbf{v}$ (each of dimension 768 corresponding to the dimensions of token embeddings), while $\mathbf{w}_j^{(O)}$ and $\mathbf{b}_i^{(O)}$ are the learned parameters of the adapter module of observation space O (also each of dimension 768). In our experiments, the length of the final embedding is 10.

The resulting observation embedding $\mathbf{h}$ is then concatenated with token embeddings of instruction, action space description, and special action tokens (as shown in in Figure A.1 (c)), while adding the positional embeddings, and passed through the controller network. The controller network is a pre-trained Distilled-GPT-2. The action embeddings are taken from the position of special action tokens, flattened and then passed through the action module. The action module of each action space has two layers of linear projection followed by ReLu non-linearity, followed by a final linear layer (with the output dimension determined by the number of actions of the corresponding action space) and a softmax layer. In our experiments, the length of action embedding is 4.

[1] huggingface's `ResNetModel.from_pretrained("microsoft/resnet-18")`

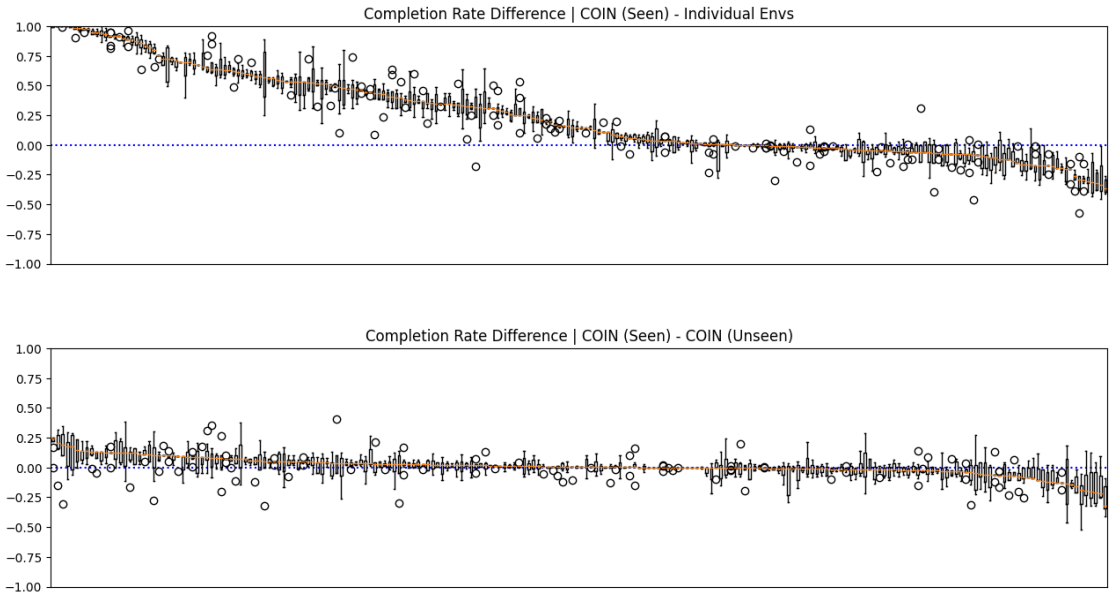


Figure A.2: Distribution of differences in performance between COIN agent on seen tasks and agent trained on individual tasks (top), as well as distribution of differences in performance between COIN agent on seen and un-seen tasks. The number of random seeds for each environment combination varies as the held-out combinations are selected by chance.

A.3 Experiments

In this section, we provide a more detailed description of the training procedure, as well as additional results. For a detailed description of the architecture, see the previous Section (A.2). In addition to the expansion of the results from Section 4.6, in A.3.3, we also add an experiment examining the ability of SOTA language models to handle compositional generalization within in-context learning.

Each model is trained for 80 epochs using AdamW optimizer and a linear learning rate schedule. The learning rate is 3×10^{-4}, with the batch size 8. We generally found that lower batch sizes result in better generalization. The completion rate of the trained agent was evaluated over 80 episodes for each environment combination (we consider the task completed if the goal state is reached in less than 100 steps), where the environment seeds used to generate training data are different from the environment seeds used in the evaluation.

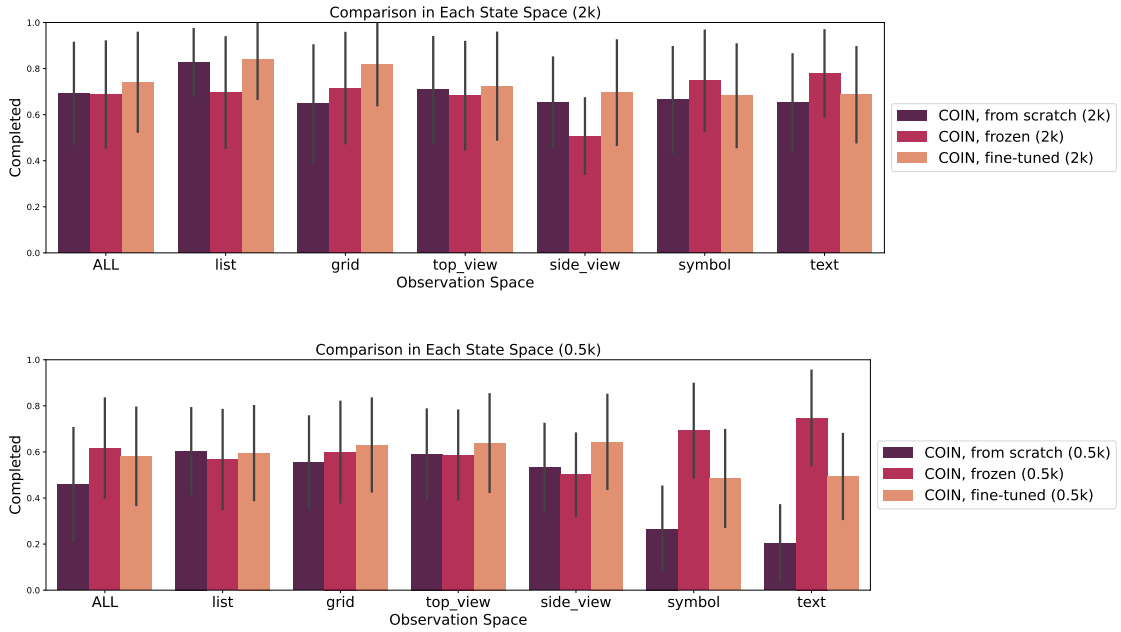


Figure A.3: Performance of COIN agent on the 40 environment combinations $\mathcal{E}^O$ containing a newly added observation space O, for each of the six available observation spaces. The controller and action modules are trained on 75% of all randomly selected combinations *not* including $\mathcal{E}^O$. In the top figure, we train on all 2048 episodes from each environment in $\mathcal{E}^O$, whereas in the bottom figure, we train on only 516 episodes from each environment. The results are reported over 3 random seeds, with the error bar representing standard deviation over all environment instances in $\mathcal{E}^O$

A.3.1 Random Holdouts

In Figure A.2, we visualized the difference in performance for each of the 240 environments and for the case of training on 75% environment combinations. Positive difference can be interpreted as the first method outperforming the second in this environment. In Figure A.2 (top), we can see that COIN agent significantly outperforms agents trained on individual combinations for a majority of environments, in some cases in fact, improving from no completed tasks to almost perfect task completion. In Figure A.2 (bottom), we compare the performance of COIN agent on seen and unseen tasks. Here we can see that difference in performance is typically within the variance of performance on individual tasks, indicating very good generalization to unseen combinations.

Figure A.4 corresponds to 4.5 (in Section 4.6.1) for the case of training on 50% of environment combinations (top) and 25% of environment combinations

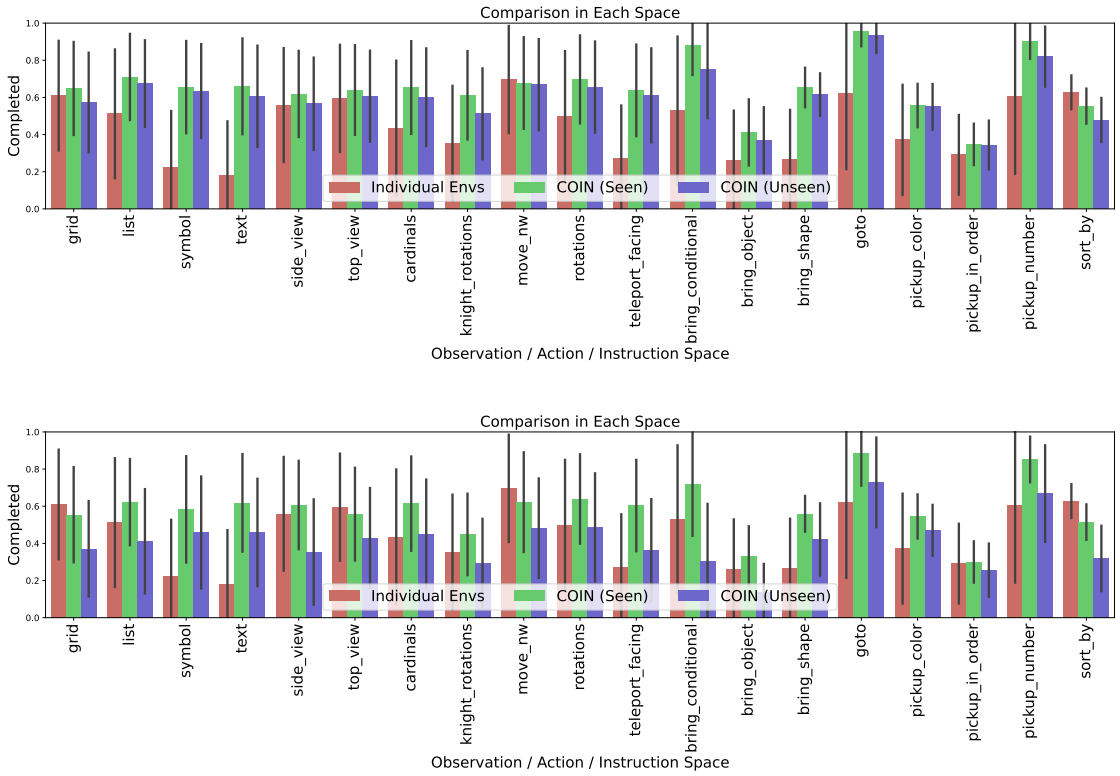


Figure A.4: Comparison of performance between individual observation, action and instruction spaces. For each space, we report the performance averaged over all environment combinations containing that space (the error bars represent standard deviation). In the top figure, we trained on 50% environment combinations, and in the bottom figure, we trained on 25% environment combinations. We report the completion rate on environment instances included (green) and *not* included (blue) in the training data. The performance of an agent trained on only one environment instance is shown in red.

(bottom). In Figure 4.3 (Section 4.6.1)„ we already demonstrated how compositional generalization overall becomes worse as the percentage of environment combinations used for training is decreased, whereas Figure A.4 demonstrates which of the spaces are affected the most.

A.3.2 New Observation Spaces

To compare the performance of COIN agent under different data sizes, we group the result presented in 4.6.3 based on the number of episodes used in training. Figure A.3 groups the results of COIN agents (either trained from scratch on the new observation space, only training the new perceptual module or training the new perceptual module alongside fine-tuning of the controller and action modules) based

on the number of episodes used in training. Figure A.3 (top) shows the results where all 2048 episodes were used in training, and Figure A.3 (bottom) shows the results where only 516 episodes were used in training. Each of the experiments is as described in Section 4.6.3. We can see here that transfer is particularly advantageous in the low-data regime and in optimizationally challenging observation spaces.

A.3.3 In-Context Learning with GPT-3

Lastly, in order to illustrate that compositional generalization with respect to observation, action and instruction spaces remains a challenge even for state-of-the-art language models, we evaluate in-context learning of the best available GPT-3 model (`text-davinci-003`) [2].

To make the task less challenging for the language model, we evaluate compositional generalization only with respect to action and instruction spaces and construct environment combinations by using: only Text observation space, a smaller set of action spaces (Cardinals, Rotations, Move NW, Knight Rotations) and easy task spaces (Go To, Pickup Number, Pickup Color, Bring Shape); for a total of 16 environment combinations. We randomly select 25% of environment combinations for evaluation in 24-shot setting. Each prompt is constructed by appending together a random selection of 24 samples from the training set, each of which is formulated as in the following example:

[2]While GPT-4 model was also available, its rate limits were too high for evaluation of completion rates. The evaluation of accuracy, however, indicates that in-context performance of GPT-4 is likely to be comparable to GPT-3, with both models achieving about 35% averaged accuracy (random guess on the same selection of action spaces has about 15% accuracy and the accuracy of COIN agent is in the 70-80% range).

"Q: The agent is at (1, 4), facing north. There is a green box at (4, 2), and a red snake at (6, 3). The agent has the following items in its inventory: yellow ball. The task is Pickup 1 item. Which of the following actions should you choose:

(a) Turn right,

(b) Turn left,

(c) Take one step forward,

(d) Pick the top item,

(e) Drop the top inventory item,

(f) Finish episode.

A: (f) Finish episode."

We construct one such prompt for each observation while acting in the held-out environment, appending the current observation to the prompt and recording the model completion. We then use this model completion to predict the next action. As before, we evaluate the method based on the completion rates in each of the held-out environments (here using 20 rollouts), and report the results over 5 random selections of the hold-out set.

The completion rate for a model using GPT-3 for action prediction was 5% (with standard deviation 6%). For comparison, the completion rates of COIN agent on the same set of environment combinations (when encountered in the random held-out set, from experiments in Section 4.6.1 with the hold-out rate 25%) is 79% (with standard deviation 18%).

B

Appendix: Diverse in Value-Relevant Features

Contents

B.1 Experiments

The additional results and the information about the experiments. For the value of hyper-parameters used in the experiments, see Table B.1.

B.1.1 Forward Mutual Information and Skill Consistency

When training agents from a fixed starting state, we observed that skills learned using reverse MI would suffer during the HRL learning phase when skills were selected by the SMDP-level policy in states which they never observed. These skills would have undefined behaviour, and in the Ant environment, would often cause the agent to "freeze", taking actions that left the agent largely static. In order to correct

Pretraining	Value
Environment Steps per Iteration	40
Environment Threads	64
GAE λ	0.99
Learning Rate	0.0001
Batches per Iteration	4
Gradient Clipping Norm	0.5
PPO Clipping Epsilon	0.2
Batch Size	1280
Learning Skills	**Value**
Learning Rate	0.0001
Environment Steps per Iteration	1000

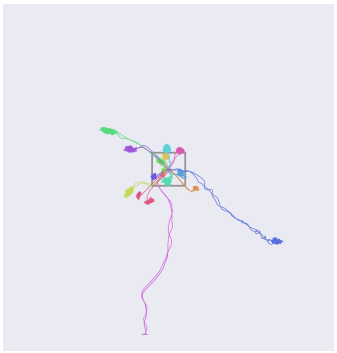
Table B.1: Table of hyper-parameters.

for this, two changes were needed: we switched to training skills on random initial states, as well as training skills using the forward mutual information objective.

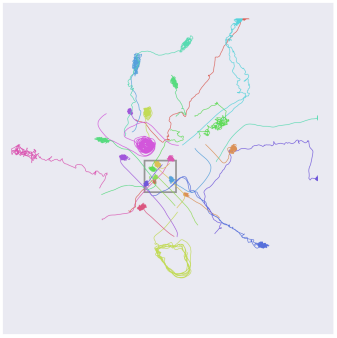
For illustrative purposes, we have examined this effect in a simple point mass environment. The agent is represented as a point in the x-y plane. It can apply a velocity in $[-1, 1]$ in each dimension independently. Because position is the only feature, applying variational skill discovery methods leads to skills that move to distinct points in the plane.

Figure B.1 demonstrates the effect of applying skills learned from a fixed initial state across randomly initialized states in the point mass environment. Skills that appear to be consistent–causing the agent to move to specific states–become ill-defined in states that they were not trained on, leading to erratic behaviour.

While this effect can be improved by training the agent on a random initial state distribution, use of the reverse mutual information objective still does not lead to skills that behave consistently. The discriminator to classifies states according to the skills they were likely to have been generated by, which can cause an implicit multi-modal partitioning of the state space, where clusters of states that correspond with a given skill are disjoint. In contrast, for the forward mutual information objective, each skill only corresponds to a single distribution over states, which can be parameterized as desired. If a unimodal distribution is chosen, then skill



(a) Skills learned using reverse MI (fixed initial state)



(b) The same skills, run on a random initial state distribution

Figure B.1: Skills learned with a fixed initial state in the point-mass environment. 16 Skills were learned using both reverse mutual information (MI) objectives. Each colour corresponds to a distinct skill. We see that skills learned using reverse MI are consistent when initialized from a fixed state, but each skill behaviour is undefined on states not visited by that skill.

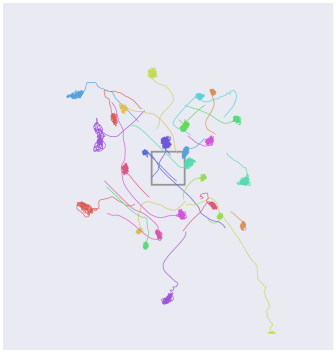
policies will be rewarded for proximity to exactly one state. This leads to consistent skills which reach specific states, regardless of the state that they are initialized from. This means that skills learned with the forward objective are useful for tasks in which skills must be composed, as consistency reduces the overall entropy of the SMDP induced by the skills.

The results of training on a random initial state distribution in the point mass environment using either reverse or forward mutual information objectives are in Figure B.2. We find that skills trained using the reverse objective still do not behave consistently, as it is easy for the discriminator to classify several sets of states as belonging to the same skill

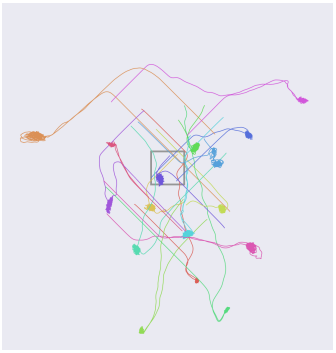
While this problem can also be potentially solved by constraining the model class of the discriminator using reverse mutual information, this may be challenging to do in abstract or complex state spaces.

B.1.2 Mujoco

In this section we expand on the details of the experiments conducted in the MuJoCo Ant domain. Following Eysenbach et al. (2019), the gear ratio of the agent is reduced to 30 in order to lead to more stable behaviour. Across all experiments in this



(a) Reverse MI (random initial state)



(b) Forward MI (random initial state)

Figure B.2: Skills learned with randomized initial states in the point-mass environment. 16 Skills were learned using both reverse and forward mutual information (MI) objectives. We see that skills learned using reverse MI can lead to single skills that are disjoint within the state space, when the agent is initialized in a random state. This does not occur when using the forward MI objective. Agents tend to move along diagonals due to a quirk in the action space: because velocity is applied in each dimension independently, moving diagonally this leads to higher overall velocity.

section, we employ multi-layer perceptron (MLP) networks with ReLU activation functions for learned functions. We employ the Adam optimizer to perform updates. The observation space includes the x-y position of the agent in order to maintain a Markov state, though we do not privilege these features in any way.

Pretraining Representations

We learn a representation for a goal seeking agent by initially training on three fixed goals. Every episode, a goal is selected from the three, and remains constant for the duration of the episode. Goal positions act as task indices here. For a visualisation of the environment layout, see Figure B.3

Our value function network is given by the goal-dependent parameterisation: $V(s, g) = f_a(\psi(s) \odot w(g))$ ψ is a MLP with two hidden layers of size 256, and an output size of 64. This relatively small output size was chosen in order to encourage compression in the learned feature space. w is a MLP with a single hidden layer of size 128, and output of 64 to perform elementwise multiplication with the output of ψ. f_a is then a final MLP with a single hidden layer of size 64, and is trained to output the value of the state for the current task.

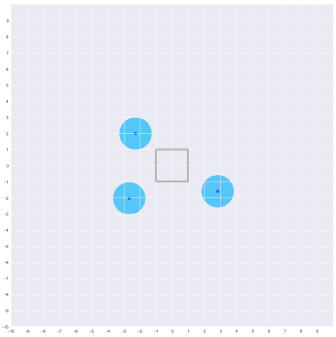


Figure B.3: Goal positions during pretraining of Ant. Dark blue dots represent goal positions, light blue circles represent the radius at which the ant receives sparse reward and the episode terminates, if the corresponding goal is active. The grey box is for scale comparison with the trace plots in the paper and here.

During this phase, we train for 20 million time steps, though this much data is not necessary. The value loss remains essentially constant after 500 000 steps. We note that it is not necessary to solve the task, or even learn the optimal value function in this phase, we only need to learn the features that correspond well with the values.

We ran this phase using five random seeds and chose the embedding to distil randomly among them. There was no observable differences between random seeds, they learned consistently, and the saliency maps of the value function were similar for all seeds.

Distillation

The distillation phase consists of training a smaller network with a tight bottleneck to predict the output of the learned embedding ψ from the state. This is done to ensure that the absolute minimal amount of information from the state is encoded in our final learned embedding, $\bar{\psi}$. The distillation was trained on states from the same distribution as the original policy was trained on. We employed an experience replay buffer of size 1 million, in order to achieve greater data efficiency during this phase.

Our distilled network was a two hidden layer MDP with 256 units in each layer, and a bottleneck size of 8. Smaller bottleneck sizes were tried, though these led to poor saliencies after learning. The output of this bottleneck was then passed through a final linear layer, giving a 64 dimensional output. The distilled network was then trained by regression of this output against the output of ϕ, using mean-squared

error loss. L1 regularisation was applied to the embedding output, and weight decay regularisation was applied to the network weights, in order to ensure the most terse and well-generalising model possible. Training was performed for 5 million time steps. Four random seeds were used, and again no significant differences were found across random seeds in terms of MSE or saliencies of learned feature maps.

Skill Learning

In this phase we learn skills that are diverse in either our learned embedding from the previous phase (for DIVRS), from the original state space (DIAYN), or a hand constructed x-y space (DIAYN + XY).

Skills were trained using SAC (Haarnoja et al., 2018) with the "double Q trick" for 3 million time steps. When learning policies and values for skills on replay buffer data, the reward function was taken from the current discriminator parameters, rather than those from when the data was collected.

For policy and value networks that condition on the active skill, a one-hot representation of the active skill is passed to a small MLP with a single hidden layer of size 64 and output dimension of 32. This embedding is concatenated with the environment observation, and passed through a three hidden-layer MLP, with the first two hidden layers of size 300, and the last of size 64, before mapping to the action space of dimension 16 for the policy network, or to a Q-value, in the case of the value network.

For reverse MI experiments, the discriminator is a two hidden layer MLP with 300 units per hidden layer. It takes in the state or embedding of state, and maps to logits for the probability that each skill is active in that state. Forward MI experiments used a lookup table corresponding to a state in the relevant space for each skill.

Five seeds were used for each skill learning procedure. Hyperparameters were selected to optimize pseudo-reward at the 3 million timestep mark. Skills were fairly consistent across seeds, though they differed somewhat in position and orientation of particular skills.

Quantitative Skill Evaluation

For quantitative evaluation of skills, we employed a sparse sequential HRL learning task, in which skills had to be composed to achieve rewards. In order to keep the state Markov, we include a one hot encoding of the number of goals reached, concatenated with the agent observation space as input to the SMDP-level policies, as well as to the flat baseline agents. Training was done for 5 million time steps. HRL agents were learned using a variant of SAC designed for discrete action spaces. This version allows for explicit computation of entropy terms, rather than the sample-based version used in continuous control.

Due to the combinatorial nature of random seeding across the several phases, for these experiments, we did not select for skills across the random seeds in the skill learning phase directly. We trained across two random seeds for 12 different hyperparameter and skillset combinations. We found that the hyperparameter ranges that we employed generally did not change learning significantly. For DIVRS, we report the average performance across hyperparameters and seeds and for baselines report the (more favorable) average of the top three most successful hyperparameter configurations, which themselves are determined by the average performance across the two seeds.

The networks used for the SMDP level policy consisted of two hidden layers of size 128, followed by a hidden layer of size 64, before outputting logits for the skill selection distribution, or for the Q values for each skill.

B.1.3 Pixel Reacher

Here we provide implementation details of the experiments performed in the Cooking Reacher environment. As was done in the Ant environment, the gear ratio of the simulated motors was lowered, this time to 80. ReLU activation functions were used for all networks, and Adam was the optimiser.

Pretraining

The pre-training procedure here is similar to the one employed in the Ant environment. First our agent is trained to reach three fixed goals in flavour space. Goal information is passed to the agent as a separate feature from the pixel state, so that our learned embedding space is not goal-dependent. Pixel information is processed via a two layer convolutional network, with a fixed kernel size of $(3, 3)$, no padding, and a stride of 2. The first layer has 64 channels, while the second has 32. From here, there is a fully connected layer of size 64. This network all together forms ϕ. The output this is elementwise multiplied by the goal embedding, which is identical to w in the Ant experiments, though now the input is in $[0, 1]^2$ to correspond with the flavour space. This phase lasts 2m frames.

Distillation

Distillation follows identically to the process described in the Ant section, but instead of an MLP, the convolutional network described in the previous section is employed. Again, a bottleneck of size 8 was used. Distillation was done for 3m frames.

Skill Learning

As in the Ant environment, the one-hot skill index is passed through a small MLP. The resulting embedding is concatenated with the output of our pixel observation CNN, which has the same architecture as the ones used in previous phases, though here the hidden layer is of size 300. In the case of the learned Q-function, the action is concatenated here as well. From here, both the critic and policy networks have a single fully connected layer of size 300, which is then mapped to either a predicted value or a distribution over actions. For the DIAYN baseline, the discriminator is identical to the network described here, though the skill id is not concatenated.

C

Appendix: Meta-Gradients In Non-Stationary Environments

Contents

C.1 BMG and $Q(\lambda)$ agent

When tuning the exploration rate ϵ of $Q(\lambda)$ agents, we use the following implementation from Flennerhag et al. (2022) to make the outer objective differentiable with respect to ϵ. In (2.31), the stochastic bootstrap policy $\pi_{\theta_K}(a|s)$ and the target policy $\pi_{\hat{\theta}}(a|s)$ are defined as:

$$\pi_{\theta_K}(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & \text{if } a = \arg\max_{a'} q_{\theta_K}(s, a') \\ \frac{\epsilon_k}{|A|} & \text{else.} \end{cases} \tag{C.1}$$

$$\pi_{\hat{\theta}}(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} q_{\hat{\theta}}(s, a') \\ 0 & \text{else.} \end{cases} \tag{C.2}$$

, where $q_{\theta_K}(s, a)$ is the learned value function, the parameters $\hat{\theta}$ are once again obtained by taking another $L - 1$ update steps, and $|A|$ is the number of different actions. The resulting objective does not require differentiation through the update rule, hence as in Flennerhag et al. (2022), we use $K = 0$.

C.2 Environments

In this section, we provide additional information about the environments used in this paper.

Two Colors

The dimension of the grid in Two Colors environment is 5×5. The observation space is constructed by concatenating one-hot encodings of x and y coordinates of positions of agents and two other objects. The total dimension of resulting observation space is $3 \times 2 \times 5 = 30$.

Switching MDPs

The reward function for each MDP is generated in the following manner: for each state-action pair there is a 50% chance of zero reward, 20% chance of reward of +1, 20% chance of reward -1, and 10% chance of a random reward sampled uniformly from the interval [-1, 1]. The transition function for each MDP is a standard two dimensional grid world—states are characterized by an (x, y) coordinate, and there are four actions that move the agent up, left, right, and down unless the intended cell is the edge of the grid or a wall is present. The transition function changes across the N MDPs by the addition of a set of walls randomly placed throughout the grid. Up to 15 walls are placed per-MDP, in sequence, by randomly sampling unoccupied cells (by goal, agent, or another wall).

At the beginning of an experiment, a set of N different MDPs is generated. At regular fixed intervals, the next grid world is sampled from this set uniformly

Inner Learner: Actor Critic	
Optimizer	SGD
Learning Rate	0.1
Batch Size	16
α_{ent} candidates *(AC only)*	$[0, 0.1, 0.2, 0.4, 0.8]$
γ	0.99
MLP hidden layers (v, π)	2
MLP feature size (v, π)	256
Activation Function	ReLU

Meta-learner	
Optimizer	Adam
ϵ (Adam)	10^{-4}
β_1, β_2 (Adam)	0.9, 0.999
Learning Rate candidates	$[10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$
$\alpha_{\text{ent}}^{(\text{outer})}$ *(MG only)*	$[0, 0.1]$
K candidates	$[1, 3, 6]$
L candidates *(BMG only)*	$[8, 16]$
H *(contextual only)*	10
MLP hidden layers *(contextual only)*	2
MLP feature size *(contextual only)*	64
Activation Function *(contextual only)*	ReLU
Output Activation	Sigmoid

Table C.1: Hyper-parameters used in experiments with Actor-Critic agents. We denote in italic when a hyper-parameter is required in only some variants of the experiments (e.g. AC baseline, MG or BMG objective, contextual meta-gradients).

with replacement. The set and the order of samples is uniquely determined by experiment random seed. If N is large compared to the total number of task switches experienced during a lifetime (e.g. $N = 1000$ with a change period $100,000$ and a lifetime of $20M$ steps), the probability of agent experiencing the same task multiple times is small. Note that since we measure the lifetime performance after $20M$ environment steps in our experiments, N does not correspond to the number of different MDPs experienced during this lifetime.

C.3 Experimental Setup and Hyper-parameters

In this section, we provide further details on the experimental setup and hyper-parameters of agents and meta-learners used in Section 6.5.

The hyper-parameters used in the experiments with AC agents are described in Table C.1. The softmax policy and the value function are implemented by two separate feed-forward MLPs. The parameters are updated every 16 environment steps: given fixed parameters, the agent interacts with the environment for 16 steps collecting observations, rewards and actions into a rollout, which is then used to compute the inner loss. The inner loss consists of the following four terms (policy, value, entropy and L_2 loss respectively):

$$\mathcal{L}^{(\text{inner})}(\theta, \mathcal{D}) = \mathcal{L}_\pi(\theta, \mathcal{D}) + \mathcal{L}_v(\theta, \mathcal{D}) + \alpha_{\text{ent}}\mathcal{L}_{\text{ent}}(\theta, \mathcal{D}) + \alpha_{\text{L2}}\mathcal{L}_{\text{L2}}(\theta). \tag{C.3}$$

In most of the experiments, the only meta-parameter is α_{ent} and $\alpha_{L2} = 0$ (i.e. there is no L_2 regularization). The exception is Section 6.5.3 (Figure 6.4b), where we tune both α_{ent} and α_{L2}. In experiments with contextual meta-gradients, we use a feed-forward MLP $g_\omega(\cdot)$ that takes context features as inputs and predicts the meta-parameter value (i.e. $\alpha_{\text{ent}} = g_\omega(c)$). When tuning two meta-parameters, we used two separate feed-forward MLPs, while taking the same context features as input (i.e. $\alpha_{\text{ent}} = g_{\omega^{(\text{ent})}}(c)$ and $\alpha_{L2} = g_{\omega^{(L2)}}(c)$). The parameters ω of these meta-networks are trained by optimizing one of the two outer losses (see Section 6.5, Experimental Axis). The output of the meta-parameter network predicting α_{L2} has been scaled by a fixed quantity (10^{-4}) to prevent training instabilities caused by too strong forgetting. The weights of both policy and value networks were included in $\mathcal{L}_{\text{L2}}(\theta)$.

The hyper-parameters used in the experiments with $Q(\lambda)$ agents are described in Table 4. The q-function is again a feed-forward MLP. The agent is optimized at each step, i.e. without batching. To avoid instabilities this could cause, we use a momentum term that maintains an exponentially moving average over gradients, with the discount factor 0.9. We sweep over the learning rate of the inner learner only for the fixed meta-parameter baseline, for the meta-gradient methods, the inner learner's learning rate is set to $3 \cdot 10^{-5}$. The tuned meta-parameter is ϵ of the ϵ-greedy exploration. The meta-parameter network is once again a feed-forward MLP, that takes context features as input and predicts ϵ (i.e. $\epsilon = g_\omega(c)$). The

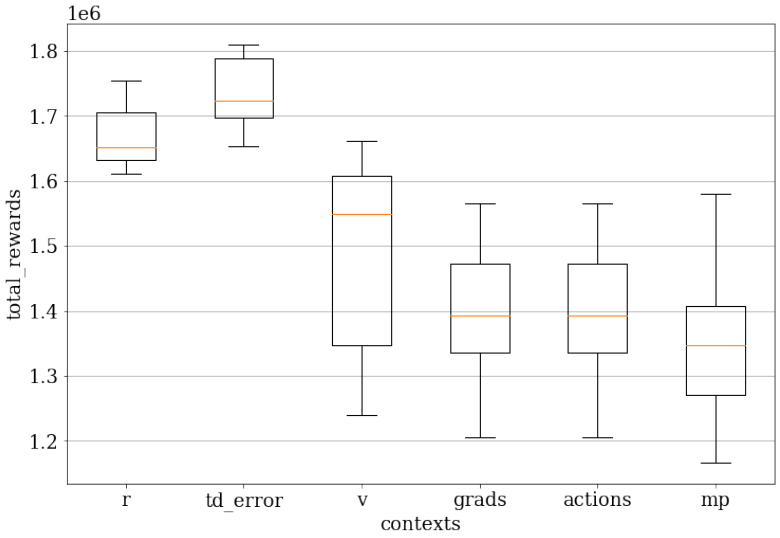


Figure C.1: Two Colors: Comparison of performance using each of the contexts individually, measured in total return after 10M steps. The agent is Actor Critic, the outer loss is BMG and we tune entropy loss coefficient. The meaning of context labels is same as in Section 6.5.3. The medians and interquartile ranges are computed over 10 random seeds.

parameters ω of this MLP are trained by optimizing the outer loss, in this case a BMG objective which has been described previously.

In experiments with both AC and $Q(\lambda)$ agents, to ensure stable initial predictions, the meta-networks were pre-trained on random context inputs sampled from uniform distribution $[-1, 1]$, to predict an output in the middle of the possible meta-parameter range (0.5 for α_{ent} and ϵ, $5 \cdot 10^{-5}$ for α_{L2}). Note that when using contextual meta-gradients, we do not sweep over the hyper-parameters introduced by the addition of context (such as the dimensions of meta-network and context history H), hence the size of the hyper-parameter sweep is the same for the meta-gradients with and without context.

C.4 Experiments

C.4.1 Context Features

We supplement the results in Section 6.5.3, Figure 6.4a, with a plot of context meta-gradients with each of the contexts individually (Figure C.1). We can observe that,

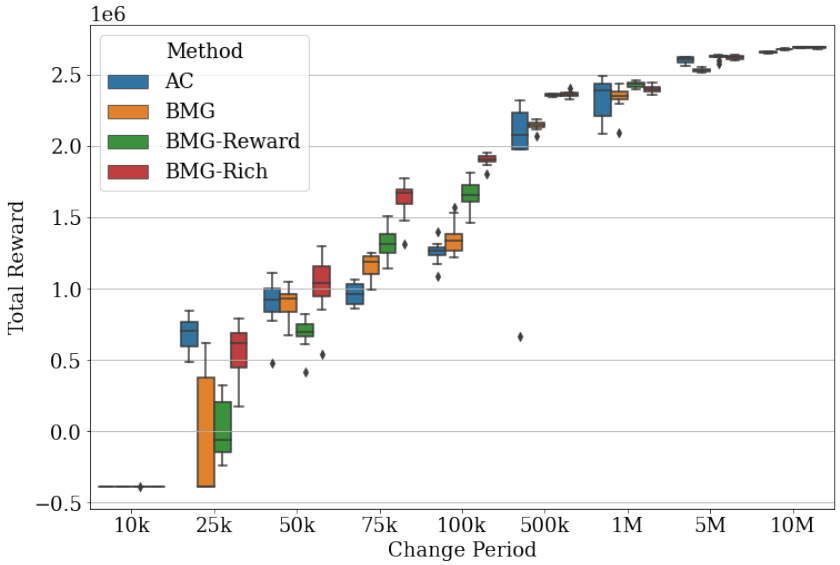


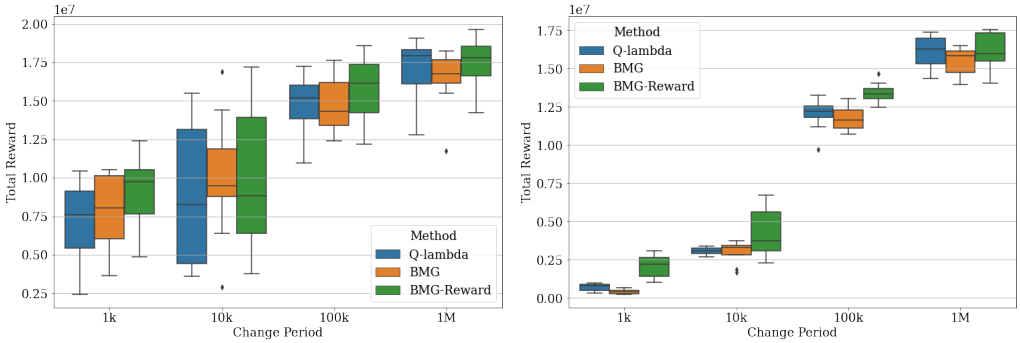
Figure C.2: Two Colors: Comparison of methods under different rates of environment non-stationarity as measured in total return after 10M steps. The medians and interquartile ranges are computed over 10 random seeds. The regime in which meta-gradients provide a significant advantage over the baseline is the greatest for meta-gradients with rich context (50k-500k steps between task switches). All meta-gradient fail to beat the baseline when the rate of non-stationarity is too high (25k steps between task switches).

looking at the each context individually, the highest gains are obtained with the contexts that highly correlate with the agent performance (reward, TD error, value).

C.4.2 Different Rates of Non-Stationarity: Total Rewards

Lastly, we supplement the results in Section 6.5.4 by illustrating how the performance of all methods drops as the rate of environment non-stationarity increases.

In Two Colors experiments (Figure C.2), no learning occurs when the environment switches every 10k steps. In Switching MDPs, note that the performance drop is much more significant when the number of different MDPs is large (Figure C.3b), indicating that in this environment, the agents benefit from repeated exposure to the same MDP.



(a) 4 MDPs: Comparison of $Q(\lambda)$, $Q(\lambda)$-BMG and $Q(\lambda)$-BMG-Reward.

(b) 1000 MDPs: Comparison of comparison of $Q(\lambda)$, $Q(\lambda)$-BMG and $Q(\lambda)$-BMG-Reward.

Figure C.3: Comparison of methods (measured in total return after 10M steps) under different rates of environment non-stationarity: (a) Switching MDPs Environment with 4 different MDPs, (b) Switching MDPs Environment with 1000 different MDPs. The performance of all methods drops when the number of different tasks is very large and as the rate of non-stationarity increases. Meta-gradients with Reward context perform the best when the rate of non-stationarity is high and the number of different tasks are high.

C.4.3 Atari Results

In Figure C.4, we report the results comparing BMG with contextual BMG on a selection of 8 Atari environments. In our experiments, we used both TD-error as well as reward, and find it leads to similar performance.

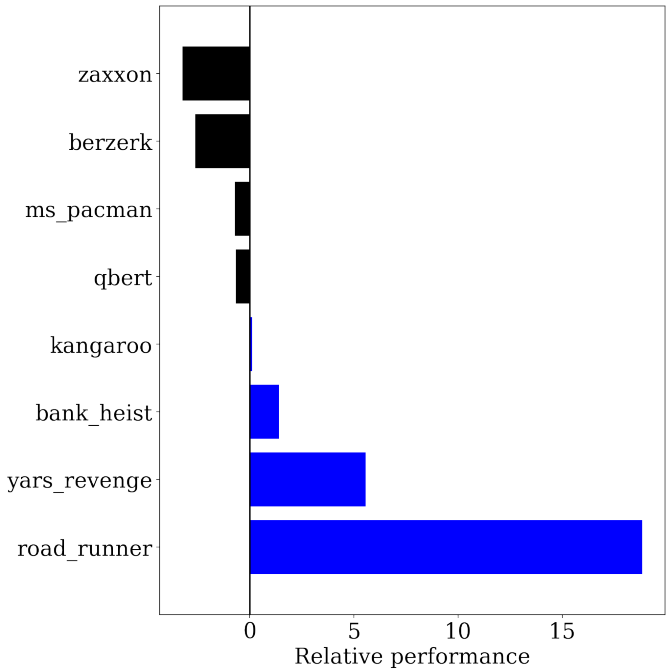


Figure C.4: Atari: Comparison of relative improvement of BMG-Reward over BMG on 8 Atari environments averaged over 3 random seeds. We find that the use of contextual information does not improve median performance on this set of tasks, and tends to help as much as it hurts performance.

Bibliography

David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pp. 2915–2923. PMLR, 2016.

David Abel, Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael L Littman, and Lawson LS Wong. State abstraction as compression in apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3134–3142, 2019.

David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, pp. 1639–1650. PMLR, 2020.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. *CoRR*, abs/1902.07198, 2019. URL `http://arxiv.org/abs/1902.07198`.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

Diogo Almeida, Clemens Winter, Jie Tang, and Wojciech Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

Jacob Andreas and Dan Klein. Alignment-based compositional semantics for instruction following. *ACL*, 2015.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017a.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular Multitask Reinforcement Learning with Policy Sketches. In *ICML*, 2017b.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. In *NAACL-HLT*, 2018.

Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to Learn by Gradient Descent by Gradient Descent. In *Advances in Neural Information Processing Systems*, 2016.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2020.

Anthropic. Claude 2, 2023. URL `https://www.anthropic.com/index/claude-2`. Accessed: 2023-09-30.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *ICCV*, 2015.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5): 469–483, 2009.

Yoav Artzi and Luke Zettlemoyer. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. In *ACL*, 2013.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Proc. Magazine*, 34(6):26–38, 2017.

Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *ICML*, 2011.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to Understand Goal Specifications by Modelling Reward. In *ICLR*, 2019.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

Stav Belogolovsky, Philip Korsunsky, Shie Mannor, Chen Tessler, and Tom Zahavy. Inverse reinforcement learning in contextual mdps. *Machine Learning*, 110(9): 2295–2334, 2021.

Yoshua Bengio. Gradient-Based Optimization of Hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. Natural language communication with robots. In *ACL*, 2016.

Daniil A Boiko, Robert MacKnight, and Gabe Gomes. Emergent autonomous scientific research capabilities of large language models. *arXiv preprint arXiv:2304.05332*, 2023.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Diana Borsa, Andre Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1VWjiRcKX`.

Abdelghani Bouziane, Djelloul Bouchiha, Noureddine Doumi, and Mimoun Malki. Question answering systems: survey and trends. *Procedia Computer Science*, 73: 366–375, 2015.

S. R. K. Branavan, Luke S Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *ACL*, 2010.

S. R. K. Branavan, David Silver, and Regina Barzilay. Learning to Win by Reading Manuals in a Monte-Carlo Framework. *JAIR*, 2012.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. *arXiv preprint arXiv:2006.12122*, 2020.

Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. *arXiv preprint arXiv:2002.03647*, 2020.

Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, July 1997. ISSN 1573-0565. doi: 10.1023/A:1007379606734. URL https://doi.org/10.1023/A:1007379606734.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-Attention Architectures for Task-Oriented Language Grounding. In *AAAI*, 2018.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017a.

David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, 2011.

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. A survey on dialogue systems: Recent advances and new frontiers. *ACM SIGKDD Explorations Newsletter*, 19(2):25–35, 2017b.

Howard Chen, Alane Suhr, Dipendra Kumar Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. *CoRR*, abs/1811.12354, 2018. URL http://arxiv.org/abs/1811.12354.

Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. In *The Eleventh International Conference on Learning Representations*, 2023.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *ICLR*, 2019.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Petros Christodoulou. Soft actor-critic for discrete action settings, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018. URL http://arxiv.org/abs/1806.11532.

Siddharth Dalmia, Dmytro Okhonko, Mike Lewis, Sergey Edunov, Shinji Watanabe, Florian Metze, Luke Zettlemoyer, and Abdelrahman Mohamed. Legonn: Building modular encoder-decoder models. *arXiv preprint arXiv:2206.03318*, 2022.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018a.

Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural Modular Control for Embodied Question Answering. *CoRL*, 2018b.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848. URL `https://ieeexplore.ieee.org/document/5206848`. ISSN: 1063-6919.

Mark A DePristo and Robert Zubek. being-in-the-world. In *AAAI*, 2001.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 2169–2176. IEEE, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, volume 98, pp. 118–126. Citeseer, 1998.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, pp. 8657–8677. PMLR, 2023.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. *arXiv preprint arXiv:2306.01324*, 2023.

Jacob Eisenstein, James Clarke, Dan Goldwasser, and Dan Roth. Reading to learn: constructing features from semantic abstracts. In *ACL*, 2009.

Anton Eklund and Mona Forsman. Topic modeling by clustering language model embeddings: Human validation on an industry dataset. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 635–643, 2022.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 201–208. JMLR Workshop and Conference Proceedings, 2010.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *ICLR*, 2019.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

Chrisantha Fernando, Dylan S. Banarse, Charles Blundell, Yori Zwols, David R Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *ArXiv*, abs/1701.08734, 2017.

John R Firth. A synopsis of linguistic theory. *Special Volume of the Philological Society*, 1957.

Sebastian Flennerhag, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell. Meta-Learning with Warped Gradient Descent. In *International Conference on Learning Representations*, 2020.

Sebastian Flennerhag, Yannick Schroecker, Tom Zahavy, Hado van Hasselt, David Silver, and Satinder Singh. Bootstrapped meta-learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=b-ny3x071E5`.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. META LEARNING SHARED HIERARCHIES. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyX0IeWAW`.

Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc Aurelio Ranzato, and Tomas Mikolov. DeViSE: A Deep Visual-Semantic Embedding Model. In *NIPS*, 2013.

Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From Language to Goals: Inverse Reinforcement Learning for Vision-Based Instruction Following. In *ICLR*, 2019.

Nancy Fulda, Daniel Ricks, Ben Murdoch, and David Wingate. What can you do with a rock? affordance extraction via word embeddings. *CoRR*, abs/1703.03429, 2017. URL `http://arxiv.org/abs/1703.03429`.

Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. In *International Conference on Learning Representations*, 2018.

Google Deepmind Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Andrea Gesmundo and Jeff Dean. munet: Evolving pretrained deep neural networks into scalable auto-tuning multitask systems. *arXiv preprint arXiv:2205.10937*, 2022.

Sandeep Goel and Manfred Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *FLAIRS conference*, pp. 346–350, 2003.

Yoav Goldberg. Assessing BERT's Syntactic Abilities. *CoRR*, abs/1901.05287, 2019.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, et al. xval: A continuous number encoding for large language models. *arXiv preprint arXiv:2310.02989*, 2023.

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3.

Alison Gopnik and Andrew Meltzoff. The development of categorization in the second year and its relation to other cognitive and linguistic developments. *Child development*, pp. 1523–1531, 1987.

Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *CVPR*, 2018.

Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019a.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. Using Natural Language for Reward Shaping in Reinforcement Learning. *CoRR*, abs/1903.02020, 2019b. URL `http://arxiv.org/abs/1903.02020`.

Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, et al. Self-supervised pretraining of visual features in the wild. *arXiv preprint arXiv:2103.01988*, 2021.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwinska, Sergio Gomez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

Significant Gravitas. Autogpt, 2023. URL `https://agpt.co`.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Austin W Hanjie, Victor Y Zhong, and Karthik Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 4051–4062. PMLR, 2021.

James Harrison, Luke Metz, and Jascha Sohl-Dickstein. A closer look at learned optimization: Stability, robustness, and inductive biases. *Advances in Neural Information Processing Systems*, 35:3758–3773, 2022.

Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic. *arXiv preprint arXiv:1902.09996*, 2019.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. pp. 2961–2969, 2017. URL `https://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html`.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3d world. *CoRR*, abs/1706.06551, 2017. URL `http://arxiv.org/abs/1706.06551`.

Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text. *arXiv preprint arXiv:2005.09382*, 2020.

Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *NIPS*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2790–2799. PMLR, May 2019. ISSN: 2640-3498.

Jeremy Howard and Sebastian Ruder. Universal Language Model Fine-tuning for Text Classification. *arXiv:1801.06146 [cs, stat]*, January 2018. URL `http://arxiv.org/abs/1801.06146`. arXiv: 1801.06146.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions. *arXiv preprint arXiv:1906.00744*, 2019.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, 2020.

Wenlong Huang, P. Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ArXiv*, abs/2201.07207, 2022a.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022b.

Wenlong Huang, F. Xia, Ted Xiao, Harris Chan, Jacky Liang, Peter R. Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2022c.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, pp. 1769–1782. PMLR, 2023.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in neural information processing systems*, pp. 13978–13990, 2019a.

Maximilian Igl, Andrew Gambardella, Nantas Nardelli, N Siddharth, Wendelin Böhmer, and Shimon Whiteson. Multitask soft option learning. *arXiv preprint arXiv:1904.01033*, 2019b.

Infocom. Zork I, 1980. URL `https://ifdb.tads.org/viewgame?id=0dbnusxunq7fw5ro`.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. Representation learning for grounded spatial reasoning. *TACL*, 2018.

Yiding Jiang, Shixiang Shane Gu, Kevin P. Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *Neural Information Processing Systems*, 2019.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2023.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, 2016.

Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pp. 752–757. Citeseer, 2005.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020. URL `https://arxiv.org/pdf/1809.02627.pdf`.

Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving Generalization in Meta Reinforcement Learning Using Learned Objectives. 2020.

Louis Kirsch, Sebastian Flennerhag, Hado Philip van Hasselt, Abram L. Friesen, Junhyuk Oh, and Yutian Chen. Introducing symmetries to black box meta reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback, 2023.

Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *HRI*, 2010.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

B. Kostka, J. Kwiecieli, J. Kowalski, and P. Rychlikowski. Text-based adventures of the golovin AI agent. In *2017 IEEE Conference on Computational Intelligence and Games*, pp. 181–188, 2017.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a Reinforcement Learner with Natural Language Advice: Initial Results in RoboCup Soccer. In *AAAI-2004 workshop on supervisory control of learning and adaptive systems*, 2004.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Igor Labutov, Bishan Yang, and Tom Mitchell. Learning to learn semantic parsers from natural language supervision. *arXiv preprint arXiv:1902.08373*, 2019.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, 2017.

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2010.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Elliott Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In *5th Annual Conference on Robot Learning*, 2021.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. *ISAIM*, 4:5, 2006.

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, et al. Pre-trained language models for interactive decision-making. *Advances in Neural Information Processing Systems*, 35:31199–31212, 2022.

Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. *arXiv preprint arXiv:2008.02790*, 2020.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=zAdUB0aCTQ`.

Joanne Lobato. Alternative perspectives on the transfer of learning: History, issues, and challenges for future research. *The journal of the learning sciences*, 15(4): 431–449, 2006.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 6309–6317. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/880. URL `https://doi.org/10.24963/ijcai.2019/880`.

Jelena Luketina, Sebastian Flennerhag, Yannick Schroecker, David Abel, Tom Zahavy, and Satinder Singh. Meta-gradients in non-stationary environments. In Sarath Chandar, Razvan Pascanu, and Doina Precup (eds.), *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pp. 886–901. PMLR, 22–24 Aug 2022. URL https://proceedings.mlr.press/v199/luketina22a.html.

Jelena Luketina, Jack Lanchantin, Sainbayar Sukhbaatar, and Arthur Szlam. Compositional interfaces for compositional generalization. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023. URL https://openreview.net/forum?id=CSTjf7dG5J.

Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023.

James MacGlashan, Monica Babes-Vroman, Marie desJardins, Michael L. Littman, Smaranda Muresan, Shawn Squire, Stefanie Tellex, Dilip Arumugam, and Lei Yang. Grounding english commands to reward functions. In *Robotics: Science and Systems XI*, 2015.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-Based Hyperparameter Optimization Through Reversible Learning. In *International conference on machine learning*, pp. 2113–2122. PMLR, 2015.

Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, 2006.

Ashique Rupam Mahmood, Richard S Sutton, Thomas Degris, and Patrick M Pilarski. Tuning-free step-size adaptation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2121–2124. IEEE, 2012.

Alana Marzoev, Samuel Madden, M Frans Kaashoek, Michael Cafarella, and Jacob Andreas. Unnatural language processing: Bridging the gap between synthetic and natural language data. *arXiv preprint arXiv:2004.13645*, 2020.

Daniela Massiceti, N Siddharth, Puneet K Dokania, and Philip HS Torr. Flipdial: A generative model for two-way visual dialogue. In *CVPR*, 2018.

Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pp. 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655681.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences. *AAAI*, 2016.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and Correcting Pathologies in the Training of Learned Optimizers. In *International Conference on Machine Learning*, 2019.

Luke Metz, C Daniel Freeman, James Harrison, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on Lifelong Learning Agents*, pp. 142–164. PMLR, 2022.

J. Mey. *Pragmatics: An Introduction*. Blackwell, 1993. ISBN 978-0-631-18689-2. URL `https://books.google.co.uk/books?id=VDGkQgAACAAJ`.

Lina Mezghani, Piotr Bojanowski, Alahari Karteek, and Sainbayar Sukhbaatar. Think before you act: Unified policy for interleaving language reasoning with actions. *ArXiv*, abs/2304.11063, 2023.

Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023a.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, 2013.

Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: a comprehensive review. *ACM computing surveys (CSUR)*, 54(3):1–40, 2021.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping Instructions and Visual Observations to Actions with Reinforcement Learning. *EMNLP*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015. doi: 10.1038/nature14236.

Aditya Modi, Nan Jiang, Satinder Singh, and Ambuj Tewari. Markov decision processes with continuous side information. In *Algorithmic Learning Theory*, pp. 597–618. PMLR, 2018.

Jesse Mu, Victor Zhong, Roberta Raileanu, Minqi Jiang, Noah Goodman, Tim Rocktäschel, and Edward Grefenstette. Improving intrinsic exploration with language abstractions. *Advances in Neural Information Processing Systems*, 35: 33947–33960, 2022.

Frank Murphy and Surya Bonaly. *Fearless Heart: An Illustrated Biography of Surya Bonaly*. Triumph Books, illustrated edition edition, January 2022. ISBN 978-1-62937-934-0.

Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018a.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *arXiv preprint arXiv:1810.01257*, 2018b.

Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, 2015.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. Grounding Language for Transfer in Deep Reinforcement Learning. *JAIR*, 2018.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Ng. Multimodal deep learning. In *The 28th International Conference on Machine Learning*. ICML, 2011.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. In *International Conference on Machine Learning*, pp. 26311–26325. PMLR, 2023.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *arXiv preprint arXiv:1707.03497*, 2017a.

Junhyuk Oh, Satinder P. Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017b.

Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering Reinforcement Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Conference on Robot Learning. Corl 2022 sim2real workshop. `https://sim2real.github.io/`, 2022. Accessed: 2023-05-17.

OpenAI. Gpt-4 technical report, 2023a.

OpenAI. Gpt-4v(ision) system card. *OpenAI.*, 2023b.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, June 2014. doi: 10.1109/CVPR.2014.222. ISSN: 1063-6919.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pp. 17156–17185. PMLR, 2022.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. Automated reinforcement learning (autorl): A survey and open problems. *arXiv preprint arXiv:2201.03916*, 2022.

Jay M Patel and Jay M Patel. Introduction to common crawl datasets. *Getting Structured Data from the Internet: Running Web Crawlers/Scrapers on a Big Data Production Scale*, pp. 277–324, 2020.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

Jing Peng and Ronald J. Williams. Incremental Multi-Step Q-Learning. In *International Conference on Machine Learning*, 1994.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, May 2018. doi: 10.1109/ICRA.2018.8460528.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018a.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *EMNLP*, 2018b.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2463–2473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL `https://aclanthology.org/D19-1250`.

Jonas Pfeiffer, Gregor Geigle, Aishwarya Kamath, Jan-Martin Steitz, Stefan Roth, Ivan Vulić, and Iryna Gurevych. xGQA: Cross-lingual visual question answering. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2497–2511, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.196. URL `https://aclanthology.org/2022.findings-acl.196`.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023.

Marc Pickett and Andrew G Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *ICML*, pp. 506–513, 2002.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, 2018.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars, and robots. In *The Twelfth International Conference on Learning Representations*, 2024.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1:9, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=1ikK0kHjvj`. Featured Certification.

Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, 2005.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

Nicol N. Schraudolph. Local Gain Adaptation in Stochastic Gradient Descent. In *International Conference on Artificial Neural Networks*, 1999.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & Compress: A scalable framework for continual learning. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4528–4537. PMLR, July 2018. ISSN: 2640-3498.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1134–1141. IEEE, 2018.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised skill discovery. *ICLR*, 2020.

Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, et al. igibson 1.0: A simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7520–7527. IEEE, 2021.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pp. 785–799. PMLR, 2023.

Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning. *ICLR*, 2018.

Anna Shusterman, Sang Ah Lee, and Elizabeth Spelke. Cognitive effects of language on human navigation. *Cognition*, 120(2):186–201, 2011.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information, 2017.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David P. Reichert, Neil Rabinowitz, André Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3191–3199, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CVPR*, 2014.

Özgür Şimşek and Andrew G Barto. Skill characterization based on betweenness. In *Advances in neural information processing systems*, pp. 1497–1504, 2009.

Ishika Singh, Gargi Singh, and Ashutosh Modi. Pre-trained language models as prior knowledge for playing text-based games. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 1729–1731, 2022.

Mark K Singley and John Robert Anderson. *The transfer of cognitive skill*. Harvard University Press, 1989.

Matthew J. A. Smith, Jelena Luketina, Kristian Hartikainen, Maximilian Igl, and Shimon Whiteson. Learning skills diverse in value-relevant features. In Sarath Chandar, Razvan Pascanu, and Doina Precup (eds.), *Proceedings of The 1st Conference on Lifelong Learning Agents*, volume 199 of *Proceedings of Machine Learning Research*, pp. 1174–1194. PMLR, 22–24 Aug 2022. URL `https://proceedings.mlr.press/v199/smith22a.html`.

Richard Socher, Milind Ganjoo, Christopher D. Manning, and Andrew Y. Ng. Zero-shot Learning Through Cross-modal Transfer. In *NIPS*, 2013.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2998–3009, October 2023.

Elizabeth Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pp. 3008–3021. Curran Associates, Inc., 2020.

Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Sean Kirmani, Brianna Zitkovich, Fei Xia, et al. Open-world object manipulation using pre-trained vision-language models. In *Conference on Robot Learning*, pp. 3397–3417. PMLR, 2023.

Sainbayar Sukhbaatar, Emily L. Denton, Arthur D. Szlam, and Rob Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *ArXiv*, abs/1811.09083, 2018.

Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing bert for aspect-based sentiment analysis via constructing auxiliary sentence. In *Proceedings of NAACL-HLT*, pp. 380–385, 2019.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999a.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 99, 1999b.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112:181–211, 1999c.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999d.

Richard S Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th international conference on Machine learning*, pp. 871–878, 2007.

Allison Tam, Neil Rabinowitz, Andrew Lampinen, Nicholas A Roy, Stephanie Chan, DJ Strouse, Jane Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *Advances in neural information processing systems*, 35:25377–25389, 2022.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4499–4509, 2017.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. In *ICLR*, 2019.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In *Advances in neural information processing systems*, pp. 385–392, 1995.

Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C. Lawrence Zitnick. ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games. *NIPS*, 2017.

Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh, and Nicolas Heess. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle, 2015.

Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 6325–6331. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/882. URL https://doi.org/10.24963/ijcai.2019/882.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Pedro Tsividis, Thomas Pouncy, Jaqueline L. Xu, Joshua B. Tenenbaum, and Samuel J. Gershman. Human learning in atari. In *AAAI*, 2017.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 2094–2100, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017b.

Vivek Veeriah, Tom Zahavy, Matteo Hessel, Zhongwen Xu, Junhyuk Oh, Iurii Kemaev, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of options via meta-learned subgoals. *Advances in Neural Information Processing Systems*, 34, 2021.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 3540–3549, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5307–5315, 2019.

Sida I Wang, Percy Liang, and Christopher D Manning. Learning Language Games through Interaction. In *ACL*, 2016.

Xin Wang, Qiuyuan Huang, Asli Çelikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. *CoRR*, abs/1811.10092, 2018. URL `http://arxiv.org/abs/1811.10092`.

Yufei Wang, Qiwei Ye, and Tie-Yan Liu. Beyond exponentially discounted sum: Automatic learning of return function. *ArXiv*, abs/1905.11591, 2019.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=KtvPdGb31Z`.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv preprint arXiv:1911.00357*, 2019.

Edward C. Williams, Nakul Gopalan, Mine Rhee, and Stefanie Tellex. Learning to Parse Natural Language to Grounded Reward Functions with Weak Supervision. In *ICRA*, 2018.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S Siegel, Jiahai Feng, Noa Korneev, Joshua B Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. *arXiv preprint arXiv:2312.08566*, 2023.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M Mitchell. Read and reap the rewards: Learning to play atari with the help of instruction manuals. *Advances in Neural Information Processing Systems*, 36, 2023.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger B. Grosse. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. In *International Conference on Learning Representations*, 2018.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.

Yiting Xie and David Richmond. Pre-training on Grayscale ImageNet Improves Medical Image Classification. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pp. 0–0, 2018.

Zhenlin Xu, Marc Niethammer, and Colin A Raffel. Compositional generalization in unsupervised compositional representation learning: A study on disentanglement and emergent language. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 25074–25087. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/9f9ecbf4062842df17ec3f4ea3ad7f54-Paper-Conference.pdf`.

Zhongwen Xu, Hado P. van Hasselt, and David Silver. Meta-Gradient Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2018.

Zhongwen Xu, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

Claudia Yan, Dipendra Kumar Misra, Andrew Bennett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. CHALET: cornell house agent learning environment. *CoRR*, abs/1801.07357, 2018. URL `http://arxiv.org/abs/1801.07357`.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8736–8754, 2020.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Haonan Yu, Haichao Zhang, and Wei Xu. Interactive Grounded Language Acquisition and Generalization in a 2d World. *ICLR*, 2018.

Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew J. Hausknecht, and Adam Trischler. Counting to explore and generalize in text-based games. *CoRR*, abs/1806.11525, 2018. URL `http://arxiv.org/abs/1806.11525`.

Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. A Self-Tuning Actor-Critic Algorithm. *Advances in Neural Information Processing Systems*, 33, 2020.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *EMNLP*, 2018.

Andy Zeng, Adrian S. Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Peter R. Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *ArXiv*, abs/2204.00598, 2022.

Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur D. Szlam. Composable planning with attributes. *ArXiv*, abs/1803.00512, 2018.

Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural crf constituency parsing. *arXiv preprint arXiv:2008.03736*, 2020.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On Learning Intrinsic Rewards for Policy Gradient Methods. *Advances in Neural Information Processing Systems*, 2018.

Allan Zhou, Vikash Kumar, Chelsea Finn, and Aravind Rajeswaran. Policy architectures for compositional generalization in control. *ArXiv*, abs/2203.05960, 2022.

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tieyan Liu. Incorporating bert into neural machine translation. In *International Conference on Learning Representations*, 2019.

Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, 2008.

George Kingsley Zipf. *Human behavior and the principle of least effort*. Addison-Wesley Press, 1949.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023.