Feature Generalization in Deep Reinforcement Learning: An Investigation into Representation Properties

by

Erfan Miahi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

In this thesis, we investigate the connection between the properties and the generalization performance of representations learned by deep reinforcement learning algorithms. Much of the earlier work on representation learning for reinforcement learning focused on designing fixed-basis architectures to achieve properties thought to be desirable, such as orthogonality and sparsity. In contrast, the idea behind deep reinforcement learning methods is that the agent designer should not encode representational properties, but rather that the data stream should determine the properties of the representation—good representations emerge under appropriate training schemes. We bring these two perspectives together, empirically investigating the properties of representations that are good at generalization in reinforcement learning. This analysis allows us to provide novel hypotheses regarding the impact of auxiliary tasks in end-to-end training of deep reinforcement learning methods. We introduce and measure six representational properties over more than 28 thousand agent-task settings. We consider DQN agents with convolutional networks in a pixel-based navigation environment. We develop a method to better understand why some representations improve generalization, through a systematic approach varying task similarity and measuring and correlating representation properties with generalization performance. Using this insight, we design two novel auxiliary losses and show that they generalize as well as our best baselines.

Preface

This thesis is based on an under-review paper titled *Investigating the Properties of Neural Network Representations in Reinforcement Learning* [82]. It is a joint work with Han Wang, Martha White, Marlos C. Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Han and I are responsible for the experiments and implementations. Martha, Adam, and Marlos wrote and edited the main part of the paper. Han and I wrote the appendix.

When I joined this project, Han had already written her master's thesis on this topic [81]. However, since I joined, this project has gone through many changes that make it sufficiently different from Han's thesis. To be specific, we significantly changed the generalization (transfer) setting, resulting in a huge increase in the number of experiments (agent-task combinations). Some of the properties in Han's thesis are removed, and some are changed, even though we refer to them by the same name in this thesis. We started recording and reporting properties during training rather than waiting until training is complete. We added a new auxiliary loss to our experiments. We started using new activation functions. We have run several new experiments.

The new contributions compared to our paper, for which I am solely responsible, are the study of the LAPLACIAN loss function, introduced in chapter 2, as an auxiliary loss and the design of two new auxiliary losses discussed in Chapter 6. We plan to extend the techniques proposed in this new chapter to more complex environments and write a new paper on it.

During my masters, I was also doing research on developing a new soft-greedy operator for exploration in reinforcement learning. This pending publication is called "Resmax: An Alternative Soft-Greedy Operator for Reinforce-

ment Learning". The results of this research were omitted from this document because it could not produce a coherent story.

To Roya

For always guiding me through difficult times!

The meaning of my existence is that life has addressed a question to me. Or, conversely, I myself am a question which is addressed to the world, and I must communicate my answer, for otherwise I am dependent upon the world's answer.

– C.G. Jung, Memories, Dreams, Reflections

Acknowledgements

No words can express how grateful I am to both of my supervisors, Martha White and Marlos Machado. I learned a lot from them simply by observing how they supervised me throughout my studies. Martha taught me to become a better writer, reviewer, and leader. Her teachings were so good that I was ranked in the top 10% ICML reviewers when I did my first official review ever (zero-shot transfer). She has this unique ability to ask thought-provoking and guiding questions throughout research projects that helped me critically think about the project, come up with new solutions, and deeply understand different aspects of the project. Despite her hectic schedule, she was always promptly available when I needed her throughout my master's program, and I am very appreciative of that. Marlos had a crucial role in helping me clearly shape and address the research questions discussed in this thesis. He consistently provided me with positive and constructive feedback, which helped me tremendously. He guided me in breaking down the research problems and highlighted their significance. I genuinely enjoyed our weekly meetings, especially witnessing his excitement about our results. He provided detailed and clear suggestions and revisions on this work, significantly improving both the thesis and my writing.

I am particularly grateful that I had the opportunity to engage in stimulating discussions and spend time with professors in the RLAI lab, namely Adam White, Rich Sutton, Csaba Szepesvari, and Rupam Mahmood. In particular, I learned many things from Rich through his class and his weekly agent-state and agent-planning meetings. Rich helped me critically think and be judicious about the research problems that I care about. These meetings and conversations were essential for motivating me to think outside the box.

Adam helped me to be a better researcher by designing concise experiments to test my research hypothesis. He has had a significant impact on how I approach empirical research in reinforcement learning.

I was really fortunate to have wonderful collaborators, namely Revan MacQueen, Abbas Masoumzadeh, Alex Ayoub, and Han Wang, on the research projects I worked on. I learned a lot and had a great time working with Han in particular. She is resilient, agile, and always cheerful. She has been one of the best researchers that I have ever worked with.

I also thank my peers, colleagues, and friends at RLAI and Amii lab for providing a stimulating and fun environment to do research. Particularly, I would like to thank Raksha Kumaraswamy, Khurram Javed, Sacha Davis, Revan MacQueen, Gábor Mihucz, Rohini Das, Aidan Bush, Zeyi Wang, Andrew Jacobson, Justin Stevens, Vlad Tkachuk, Shibhansh Dohare, Prabhat Nagarajan, Samuel Neumann, Farzane Aminmansour, Alex Ayoub, Kirby Banman, Andy Patterson, Chunlok Lo, David Tao, Prabhat Nagarajan, Sina Ghiassian, Abhishek Naik, Matthew Schlegel, Yi Wan, Ehsan Imani, and Han Wang. Khurram provided me with valuable guidance at the beginning of my studies. I also appreciate the way he challenged my ideas during our discussions. It has been a great pleasure to work in this research-friendly group.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the reinforcement learning problem an agent interacts with its environment, receiving observations and taking actions based on those observations, with the goal of maximizing the sum of a special numerical signal, the reward. In this context, the first problem an agent faces is the problem of *agent state* construction: to determine how to process observations to summarize the state they are in. The function that converts these observations is known as *representation*, its elements are known as *features*, and the process of learning such function is known as *representation learning*. Ultimately, many other subproblems depend on the successful construction of agent states. Bad representations hinder predictions and diminish the effectiveness of planning and learning algorithms [13, 27, 76, 80]. Good representations can lead to better sample efficiency [40, 64]. Therefore, the key question is: *what are good representations and how can the agent find them?*

The goodness of a representation is often defined by its ability to generalize properly over the state space [74]. The act of generalization itself can be either good or bad. Bad generalization can entirely disrupt the learning process. Good generalization ultimately should serve faster learning (See Figure 1.1 as an example). We consider the view toward generalization that draws a clear line between training and evaluation procedures [cf. 32].[1] It considers training the agent on a distribution (set) of tasks, called training environments, and

[1] Of course, generalizing over the state space in a single environment is equally important, especially when training in a continual setting. However, here we focus on the multiple-environment definition of the generalization.

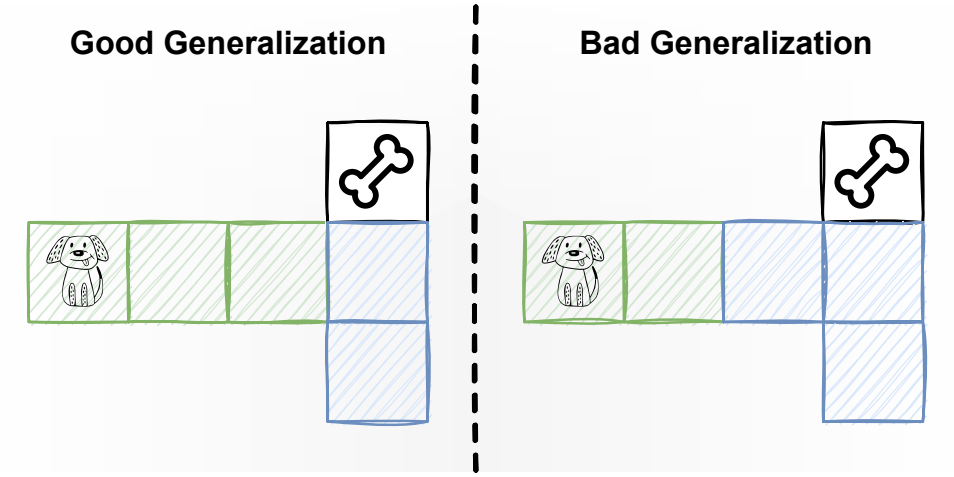

Figure 1.1: Good versus bad generalization in a simple T-maze environment. In this environment, the agent is a dog with the goal of reaching the bone. The agent can take four actions given the representation of a cell: moving up, down, left, and right. Taking actions toward walls has no effect. The representation generalizes over the cells (i.e., states) with the same colors. This means that cells with the same color look the same to the dog. Consequently, the dog chooses the same action in these cells. In the left maze, the dog can learn to reach the bone pretty quickly by just learning to go right in the green states and to go up in the blue states. The generalization in the right maze, on the other hand, is rather poor, making it difficult for the dog to approach the bone.

then evaluating the trained agent on another distribution of tasks called testing environments. In this setting, we consider good generalization to facilitate learning on the testing environments.

Fixed transformations of the agent's observations, which lead to fixed-basis architectures, have been extensively explored in reinforcement learning. They allow us to enforce specific properties that are thought to be beneficial. For example, many approaches either use or search for orthogonal or decorrelated features, such as orthogonal matching pursuit [54], Bellman-error basis functions [57], Fourier basis [34], tile coding [73], and proto-value functions [33, 49]. Prototypical input matching methods have been explored, as in kernel methods, radial basis functions [74], cascade correlation networks [14], and Kanerva coding [30]. Most of the above representations project the input to a low-dimensional space to encourage the representation to encode

only the most important information, saving memory and computational resources. High-dimensional, sparse representations have also been proposed, as they are more likely to be orthogonal. Moreover, by activating only a small subset of features, a sparse representation reduces computation and increases scalability, such as in tile coding [73], and in sparse distributed memories [61]. However, fixed-basis architectures are not adaptive and they are difficult to scale to high-dimensional input spaces.

Recent developments in representation learning for reinforcement learning explore a different perspective: we should avoid optimizing specific *properties*[2] and instead use gradient descent to let the training data dictate the properties of the representation. This view is widely held, and is reflected in the focus on specifying training regimes, including using multi-task (parallel) training [7, 19, 77], auxiliary losses [5, 28], and training on a distribution of problems (à la meta-learning) [16, 29, 52, 65, 67]. The basic idea underlying all these approaches is that good representations will emerge if the problem setting is complex enough. This view gave birth to a new branch of algorithms called *deep reinforcement learning* [50].

In this thesis, we study the problem of generalization in deep reinforcement learning, which aims to improve sample efficiency of an agent in a predetermined set of tasks. Although, many deep reinforcement learning algorithms have been developed to address this problem, little work has been done to understand and evaluate the generalization capability of the representations that emerge through these algorithms. The most common approach is to visualize the learned representations [5, 11, 17, 18, 24, 25, 35, 50, 63, 69, 72, 84, 92], which is mainly adopted from the supervised learning literature. However, in reinforcement learning, the impact of delayed consequences and temporally correlated data makes it difficult to import analysis techniques from other fields, and recent work highlighted how popular approaches like saliency maps may not be totally appropriate [3]. This work takes a different approach for

[2]There is, of course, work in reinforcement learning exploring how to encode specific properties on the network, such as sparse activations [44, 55], disentangled features [26], and orthogonality constraints [18, 88].

evaluating generalization. Specifically, we explore the properties of representations learned by modern deep reinforcement learning systems in a setting where representations are re-used in similar but unseen tasks, and we study how these properties correlate with the generalization performance of these representations.

1.1 Thesis Statement

The central claim of this work is that **throughout the training phase, several properties that can explain generalization performance will emerge in the representations learned by deep reinforcement learning algorithms.**

Deep reinforcement learning algorithms enable the agent to learn representations by using neural networks to extract useful features from the inputs. Each layer of a neural network produces its output often using a non-linear function known as the *activation function*.

As mentioned above, *representation* is a vector resulting from the transformation of the input states. Its main purpose is to capture useful information that can facilitate learning. In this work, we consider the representation to be the output of a *representation function*, which is a trainable model that transforms the input states. This definition makes it easy to design properties that can be measured from the representation.

Properties summarize the representation. We investigate properties grouped into three categories: *capacity* (complexity reduction, dynamics awareness, and diversity), *redundancy* (orthogonality and sparsity), and *update robustness* (non-interference).

1.2 Approach

We support the thesis statement by empirically studying the properties of representations learned by deep reinforcement learning algorithms. Specifically, we consider Q-learning with neural network function approximation [50] coupled with auxiliary tasks (i.e., additional prediction tasks that are thought to

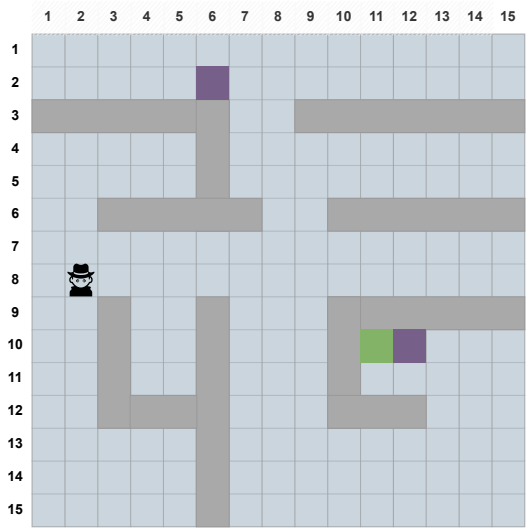

Figure 1.2: Maze problem. The position of the walls (dark grey), the goal (green) in the training, and two testing tasks (purple) are shown.

improve the representation). We focus on a specific generalization setting: one with a training phase to learn a representation, followed by a testing phase in a variety of tasks with similar dynamics but different rewards.

We carefully design this generalization setting to help us address the main thesis of this work. To do so, we consider a simple pixel-based navigation environment (shown in Figure 1.2), where successful generalization is challenging but possible. This environment can be readily used to generate numerous related tasks. We propose a technique to generate testing environments based on their similarity to the training environment, resulting in 173 testing environments. This approach helps us to analyze how representations generalize as the testing tasks become more dissimilar.

This setting has two phases: a training phase and a testing phase. During the training phase, we train our representations with different activation functions and auxiliary tasks. We ensure that the agent is capable of solving the training tasks. As a result, we end up with representations that vary in their properties and generalization performance, which is crucial for further analysis in this thesis. Then, we fix (freeze) the parameters of the representation function and evaluate its performance on the testing environments. Fixing these parameters is critical to our study because it results in having fixed rep-

resentation properties during the testing procedure, allowing us to carefully evaluate our thesis statement.

Our experiments generate a mountain of data that require further analysis. We introduce a variety of data visualization techniques to understand the connection between the representation properties and generalization performance. Specifically, we use these techniques to analyze the non-linear correlation, further helping us recognize which representation properties correlate to good generalization performance.

In the final step of this study, we explicitly maximize the properties that are shown to improve the performance in the representation to further evaluate the thesis of this work. To do so, we introduce a set of auxiliary losses that maximize these representation properties. We first investigate how incorporating these auxiliary losses affects representation properties. Then, we show that using them indeed improves the generalization performance of the representations to the point that they can compete with our best baselines.

1.3 Contributions

The key contributions of this dissertation are:

Methodology. We introduce new methodologies to evaluate representations in deep reinforcement learning. These methodologies consist of three components: representation properties, task similarity metric, and several approaches to analyze the data. We study six properties consisting of both a subset of properties discussed in the literature and properties newly introduced in this work (Chapter 3). We measure these properties in more than $28,000$ agent-task settings and show that they correlate with generalization performance (Chapter 5). We further advocate for the importance of these properties by showing that optimizing them leads to good representations (Chapter 6). We design a technique to measure the similarity between any set of environments and use it to analyze generalization performance across the 173 tasks that are ranked based on their similarity (Chapter 4). We provide several mechanisms to aggregate and visualize the mountain of data produced

across representations. We use them in this thesis to study and understand the relevance of representation properties to generalization performance. This set of techniques can be used as a general framework for others to analyze representations (Chapter 5).

Auxiliary losses. The role and significance of auxiliary losses in reinforcement learning remain poorly understood. We explore the representations learned using different variations of eight auxiliary losses.We particularly analyze this family of losses by performing an ablation study to determine which components of these losses are critical for improvement in generalization performance (Chapter 5). We further design new auxiliary losses based on the insights that we gained from our analysis, and show that these losses can match the generalization performance of our best baselines (Chapter 6).

1.4 Dissertation layout

This document is divided into seven chapters. Chapter 2 introduces the reinforcement learning problem, as well as an overview of generalization and representation learning in reinforcement learning. Chapter 3 discusses the representation properties and approaches to measure them. The setting for evaluating generalization is introduced in Chapter 4. Chapter 5 presents the experimental study carried out in this work and makes a connection between the representation properties and the generalization performance. In Chapter 6, we propose and evaluate a set of auxiliary losses to maximize promising properties. Finally, we summarize the contributions of this work and discuss potential future research directions in Chapter 7.

Chapter 2

Background & Problem Formulation

In this chapter, we introduce the reinforcement learning problem, deep reinforcement learning solution methods, as well as the concept of generalization in reinforcement learning. We provide a precise definition of representation in deep reinforcement learning and discuss different approaches for learning a good representation. Building on this, we review relevant work on generalization in deep reinforcement learning and explain how this work fits within it.

Throughout this dissertation, as a convention, we will indicate matrices by bold capital letters (e.g., $\boldsymbol{S}$), random variables by capital letters (e.g., S_t, R_t), vectors by bold lowercase letters (e.g., $\boldsymbol{\theta}$, $\boldsymbol{\phi}$), functions and scalars by non-bold lowercase letters (e.g., ϕ, q), and sets with a calligraphic font (e.g., $\mathcal{S}$, $\mathcal{A}$). Subscripts are only used for naming purposes and do not provide any information about the letter's type.

2.1 Problem Formulation and Notation

We formalize the agent's interaction with the environment as a finite Markov Decision Process (MDP) with a finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and bounded reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. On each time step, $t = 1, 2, ...$, the agent takes action A_t in state S_t and the environment transitions to state $S_{t+1} \sim p(\cdot|S_t, A_t)$ and emits

a reward R_{t+1}. The agent's objective is to find a policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes the expected discounted sum of future rewards, the *return*, $G_t \doteq R_{t+1} + \gamma_{t+1} G_{t+1}$, where $\gamma_{t+1} \in [0, 1]$ denotes a discount that depends on the transition (S_t, A_t, S_{t+1}) [85].

We focus on value-based techniques, in which the agent attempts to learn the optimal action-value function and uses it to shape its policy. The action-value function for each state-action pair under policy π is

$$q_\pi(s, a) = \mathop{\mathbb{E}}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]. \tag{2.1}$$

We can use tabular techniques to iteratively learn these action-values through agent-environment interactions. These approaches use a table to approximate action-values, with each table cell representing the estimated value of a distinct pair of states and actions. Q-learning is one such technique that iteratively updates this table using:

$$q(s, a) \leftarrow q(s, a) + \alpha \left(r + \gamma \max_{a'} q(s', a') - q(s, a) \right),$$

where s' is the next state.

To learn these action-values, the agent needs to explore. One simple strategy to promote exploratory behavior is ε-greedy. Using this technique, the agent selects the greedy action with respect to its action-value with probability ε and a random action with probability $1-\varepsilon$. We use this simple exploration strategy in this thesis.

2.2 Function Approximation

The state space is often too large to be represented in a table. To overcome this issue, we need to generalize across states. We can achieve this by transforming the state space to the representation space using a representation function $\phi : \mathcal{S} \to \mathbb{R}^d$. A good representation function should help the agent perform well with less memory, computation, and with fewer interaction with the environment.

We can use, for example, Q-learning with linear function approximation to train the agent with a given handcrafted representation. This algorithm approximates the action-value function by linearly mapping the representation to action-values, with a value function $\hat{q}_{\boldsymbol{\theta}}$ (parameterized by $\boldsymbol{\theta}$). This algorithm updates its parameters, using the update:

$$\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} + \alpha \left(r + \gamma \text{max}_{a'} \hat{q}_{\boldsymbol{\theta}}(\phi(s'), a') - \hat{q}_{\boldsymbol{\theta}}(\phi(s), a)\right) \nabla \hat{q}_{\boldsymbol{\theta}}(\phi(s), a). \tag{2.2}$$

Designing a good representation function requires a great deal of expert knowledge and, in some cases, it is impractical due to the complexity of the environment. We can learn a good representation function instead. Deep reinforcement learning algorithms are the most successful approach for that. The word *deep*, in deep reinforcement learning, stems from the use of neural networks to approximate q_π and ϕ. Neural networks are made up of a series of layers. Each layer of a neural network typically performs a linear transformation on its input before passing it to a non-linear function known as the activation function.

A successful extension of Q-learning to deep reinforcement learning setting is DQN [50]. In DQN, the approximate value function is learned by a neural network that is parameterized by a set of weights $\boldsymbol{\theta}$: $\hat{q}_{\boldsymbol{\theta}}(s, a) \approx q_\pi(s, a)$. In the original paper, this neural network consists of three convolutional layers followed by a single fully connected layer, as shown in Figure 2.1. During training, DQN iteratively updates its action-value estimates by training the parameters of a neural network, $\boldsymbol{\theta}$, with stochastic gradient descent: $\Delta\boldsymbol{\theta} \propto \big(R_{t+1} + \gamma_{t+1} \max \hat{q}_{\bar{\boldsymbol{\theta}}}(S_{t+1}, a) - \hat{q}_{\boldsymbol{\theta}}(S_t, A_t)\big) \nabla_\theta \hat{q}_{\boldsymbol{\theta}}(S_t, A_t)$. The target network, $\hat{q}_{\bar{\boldsymbol{\theta}}}$, is not updated on every step, but only periodically set equal to the current $\hat{q}_{\boldsymbol{\theta}}$. Actions are selected according to an ϵ-greedy policy. DQN samples a single mini-batch from an experience replay buffer, $\mathcal{D}$, to update the value function. In this work, we focus on the representations learned by this algorithm.

We use a unified architecture to explore representations induced by a variety of auxiliary tasks, as shown in Figure 2.2. The first layers, parameterized by $\boldsymbol{\theta}_R$, produce the representation $\boldsymbol{\phi}_t = \phi_{\boldsymbol{\theta_R}}(s_t)$. The value network uses this

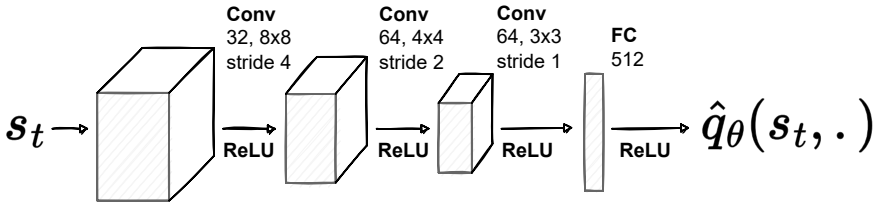

Figure 2.1: The original architecture of DQN network.

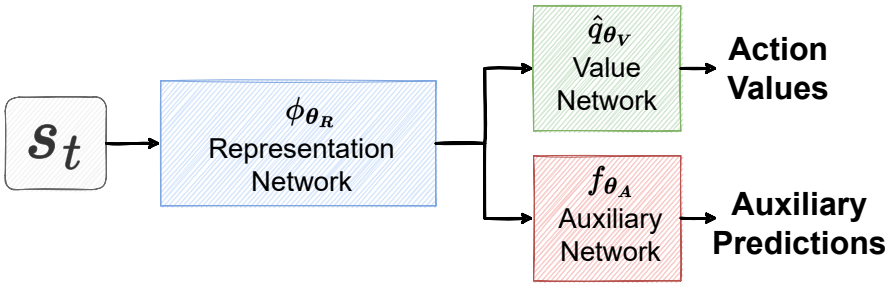

Figure 2.2: The unified architecture used in this work. The representation network $\phi_{\boldsymbol{\theta_R}}$ learns a mapping from input-state s_t to the agent-state (representation of s_t). The representation network is learned to improve two objectives: performance on a main task, and on an auxiliary task. Our agents only use one auxiliary task at a time.

representation to compute the action-values. The auxiliary tasks are encoded with additional layers and separate heads ($f_{\boldsymbol{\theta_A}}$ with parameters $\boldsymbol{\theta_A}$), further impacting the updates to $\boldsymbol{\theta}_R$ via gradient descent. Given this architecture, $\phi_{\boldsymbol{\theta_R}}(s)$ must be adjusted to be useful both for estimating action values and for reducing the auxiliary losses.

2.3 Representation Learning

There are three common approaches to improve the representations learned by deep reinforcement learning algorithms. First, we can make specific transformations to the input states before feeding them to the neural network [e.g., 1, 38, 39, 71, 91]. These transformations help the agent to be robust to certain changes in the input states. They are known as data augmentation techniques and often applied to pixel-based input states. The second approach involves modifying the neural network architecture, such as adjusting its activation function [e.g., 37, 55, 68, 83]. The third approach is to encourage the agent to

learn additional prediction tasks that incentivize the network to learn about properties of the environment which are, in principle, not directly related to reward maximization. These tasks are called auxiliary tasks [e.g., 5, 28, 51, 71]. The remainder of this section discusses the key approaches employed in our study.

2.3.1 Data Augmentation Techniques

Deep reinforcement learning research has just recently started to investigate the efficacy of data augmentation techniques, such as rotating the input image and flipping it horizontally or vertically. One important study investigates the effectiveness of different data augmentation techniques and concludes that random shift (randomly shift the input image) is the most effective technique [91].

2.3.2 Activation Functions

In this work, we explore two different activation functions: ReLU and Fuzzy Tiling Activation (FTA) [55]. ReLU is a standard activation function, defined as $\max(z, 0)$ for input z, where z is a linear weighting on the previous layer. FTA is a newly introduced activation, designed to generate sparse outputs. Essentially, it bins the scalar input into k bins, with some smoothing to ensure non-zero gradients through the activation. The smoothness and bin width are controlled by a parameter $\eta > 0$. The interval is from $[-\frac{\eta}{2}k, \frac{\eta}{2}k]$, with k equally sized bins of size η. Assume the input z is in bin i, namely $-\frac{\eta}{2}k + (i-1)\eta \leq z \leq -\frac{\eta}{2}k + i\eta$ where $i \in \{1, 2, \ldots, k\}$. The k-dimensional output vector $\mathbf{h}(z)$ given by FTA on z has entries $h_j(z) \in [0, 1]$ defined as

$$h_j(z) = \begin{cases} 1 & \text{if } j = i, \\ 1 + \eta(j - \frac{k}{2}) - z & \text{if } j < i \text{ and } z < \eta(1 + j - \frac{k}{2}), \\ 1 + z - \eta(j - 1 - \frac{k}{2}) & \text{if } j > i, z > \eta(j - 1 - \frac{k}{2}), \\ 0 & \text{else.} \end{cases}$$

Larger values of η activate more entries in $\mathbf{h}(z)$, and smaller values of η result in more sparsity. We use the suggested default choice of $\eta = \delta$, where δ determines the bin size [cf. 55].

2.3.3 Auxiliary Tasks

Here, we present the auxiliary tasks that are used in this work. Auxiliary losses of these tasks are computed using the transitions stored in the replay buffer, $\mathcal{D}$. However, each loss function samples from the replay buffer in a different way. We go through the specifics of the auxiliary network and implementation of auxiliary tasks in Chapter 5. The explanation and motivation of these loss functions are outlined below.

Input Reconstruction (IR): This auxiliary task tries to reconstruct the network's input, as in an autoencoder. This extraction is achieved by using a bottleneck layer: a low-dimensional layer that forces only the most important information to be retained and the remainder, including the noise, to be discarded. We include this auxiliary task as a classic and simple choice:

$$\mathcal{L}_{\text{IR}}\left(\mathcal{D}\right) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[\|f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s)) - s\|_2^2\right]. \tag{2.3}$$

Next Agent State Prediction (NAS): Another common choice is to predict the next agent state (i.e., next state's representation) [8, 18, 28, 48, 52, 59, 65, 69, 89]. This loss encourages the representation to capture the transition dynamics. The agent predicts $\boldsymbol{\phi}_{t+1}$ using $\boldsymbol{\phi}_t$ and action a_t. Predicting the next agent state might give vacuous solutions when it is the only training signal; however, jointly training with the main task prevents this from happening. The combination of this auxiliary loss with the main task encourages the representation to both be useful for action-value estimation, as well as capable of anticipating features on the next step. Several papers have highlighted that the ability to predict the next state is related to the ability to predict action-values [5, 56, 75]. We consider the following loss to represent NAS:

$$\begin{aligned}\mathcal{L}_{\text{NAS}}\left(\mathcal{D}\right) = \mathop{\mathbb{E}}_{\substack{(s_t,a_t,s_{t+1})\sim\mathcal{D}\\(s_{k\neq t},a_{k\neq t})\sim\mathcal{D}}}\Big[&\|f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_t), a_t) - (\phi_{\boldsymbol{\theta_R}}(s_{t+1}) - \phi_{\boldsymbol{\theta_R}}(s_t))\|_2^2 \\ &+ \max\left(0, 1 - \|f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_k), s_k) - (\phi_{\boldsymbol{\theta_R}}(s_{t+1}) - \phi_{\boldsymbol{\theta_R}}(s_t))\|_2^2\right)\Big].\end{aligned} \tag{2.4}$$

Successor Feature Prediction (SF): NAS can be taken one step further, with the target including not just the next agent-state but many future

agent states [47]. *Successor features* (SFs) provide just such a target. SFs are defined with respect to a particular policy π as $\psi_t^\pi = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i \boldsymbol{\phi}_{t+i} \right]$. They have been used in the transfer setting because they can be used to quickly infer value estimates for new reward functions that are a linear function of $\boldsymbol{\phi}_t$ [4]. In the tabular case, SFs correspond to the successor representation [12], which is equivalent to proto-value functions [48]. We opt to use the greedy policy according to the action values for the training task, which means the SFs are tracking a changing policy. This loss is

$$\begin{aligned} \mathcal{L}_{\text{SF}}\left(\mathcal{D}\right) = \mathop{\mathbb{E}}_{(s_t,a_t,s_{t+1},a_{t+1})\sim\mathcal{D}} \Big[\| f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_t), a_t) - \\ (\phi_{\boldsymbol{\theta_R}}(s_t) + \gamma_{\text{SF}} f_{\bar{\boldsymbol{\theta}}_{\boldsymbol{A}}}(\phi_{\bar{\boldsymbol{\theta}}_{\boldsymbol{R}}}(s_{t+1}), a_{t+1})) \|_2^2 \Big], \end{aligned} \tag{2.5}$$

where $\gamma_{\text{SF}} \in [0, 1]$.

Reward Prediction (Reward): Another auxiliary task we consider is predicting the one-step reward in the future based on the current state and action [28]. The prediction requires the agent to encode the reward information that it can obtain in the short term in the representation function. This can be simply captured by the following loss:

$$\mathcal{L}_{\text{Reward}}\left(\mathcal{D}\right) = \mathop{\mathbb{E}}_{(s_t,a_t,r_{t+1})\sim\mathcal{D}} \Big[\| f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_t), a_t) - r_{t+1} \|_2^2 \Big]. \tag{2.6}$$

Expert Target Prediction (XY): Another auxiliary task is the prediction of expert-designed targets. It is based on the idea that a good representation should be able to predict key artifacts of an environment. This requires domain knowledge and is not always possible. Here we consider the coordinates of the agent in our navigation task as the target predictions. We consider the following loss

$$\mathcal{L}_{\text{XY}}\left(\mathcal{D}\right) = \mathop{\mathbb{E}}_{(s_t,x_t,y_t)\sim\mathcal{D}} \left[\| f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_t)) - [x_t, y_t] \|_2^2 \right], \tag{2.7}$$

where x_t is the row number and y_t is the column number of the agent's location in the maze.

Virtual Value Function Learning (VirtualVF): This auxiliary task is based on the tasks the agent will face in the testing phase. We consider one

auxiliary loss that uses a goal location at the center of the maze (VIRTUALVF-1), and another that uses five goals at the four corners and the center of the maze (VIRTUALVF-5). These are virtual tasks because the agent imagines achieving these goals, even though they are not the training goal. We use VIRTUALVF-1 and VIRTUALVF-5 to assess the utility of having a larger set of virtual goals. We learn these auxiliary value functions with DQN. We define this loss as

$$\mathcal{L}_{\text{VirtualVF}}(\mathcal{D}) = \sum_{g \in \mathcal{G}} \mathop{\mathbb{E}}_{(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}) \sim \mathcal{D}} \Big[\| f^g_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(s_t), a_t) - (r^g_t + \gamma_{\text{VF}} f^g_{\bar{\boldsymbol{\theta}}_{\boldsymbol{A}}}(\phi_{\bar{\boldsymbol{\theta}}_{\boldsymbol{R}}}(s_{t+1}), a_{t+1}) \|^2_2 \Big], \tag{2.8}$$

where $\phi_{\bar{\boldsymbol{\theta}}_{\boldsymbol{R}}}(s_{t+1})$ is the representation function of the target network, $\gamma_{\text{VF}} \in [0, 1]$, and $\mathcal{G}$ is a set of goal locations. The reward r_g, value function, $f^g_{\boldsymbol{\theta_A}}$, and target value function, $f^g_{\bar{\boldsymbol{\theta}}_{\boldsymbol{A}}}$, are associated with the goal g.

Augmented Temporal Contrast (ATC): This contrastive loss encourages the network to learn similar representations for input-states that are temporally close to each other. This auxiliary task led to the first successful pre-training of a deep reinforcement learning agent, meaning it led to representations that could be generally reused for other tasks. ATC also includes other augmentations. Specifically, this loss uses a data augmentation function, AUG, an additional target representation and target auxiliary network ($\phi_{\breve{\boldsymbol{\theta}}_{\boldsymbol{R}}}$ and $f_{\breve{\boldsymbol{\theta}}_{\boldsymbol{A}}}$), as well as an additional auxiliary network $f_{\boldsymbol{\theta_C}}$ [cf. 71]. We test it with these additions, to report performance of the originally proposed approach, even though it goes beyond strictly only adding an auxiliary loss. This loss is presented below

$$\mathcal{L}_{\text{ATC}}(\mathcal{D}) = -\mathbb{E}_{\mathcal{D}} \left[\log \frac{\exp(p_i \boldsymbol{W} c_{i+k})}{\sum_{s_j \in \mathcal{S}} \exp(p_j \boldsymbol{W} c_{j+k})} \right], \tag{2.9}$$

where $\boldsymbol{W}$ is a matrix of trainable weights, $p_t = f_{\theta_C}(f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(\text{AUG}(s_t)))) + f_{\boldsymbol{\theta_A}}(\phi_{\boldsymbol{\theta_R}}(\text{AUG}(s_t)))$, and $c_{t+k} = f_{\breve{\boldsymbol{\theta}}_{\boldsymbol{A}}}(\phi_{\breve{\boldsymbol{\theta}}_{\boldsymbol{R}}}(\text{AUG}(s_{t+k})))$.

LAPLACIAN: Lastly, we employ a recently introduced Laplacian loss function that captures the geometry of the underlying state space (i.e., the environment's dynamics) [88]. This loss is the first efficient and successful approximation of the eigenvectors of the Laplacian in the reinforcement learning.

Similar to ATC, it encourages the representation of the states that are dynamically close to each other to be similar, and those that are widely apart to be distinct. However, whereas ATC encourages the representations of state that are k timestep apart from each other to be similar, this loss encourages each state's representation to be similar to the representation of all its future states. It also uses a completely different auxiliary network architecture and loss to achieve this. We compute this loss using

$$\begin{aligned}\mathcal{L}_{\text{LAPLACIAN}}(\mathcal{D}) = &\mathop{\mathbb{E}}_{(s_t, s_{t+\bar{k}})\sim\mathcal{D}}\left[\|\phi_{\boldsymbol{\theta_R}}(s_t) - \phi_{\boldsymbol{\theta_R}}(s_{t+\hat{k}})\|_2^2\right] \\ +\beta &\mathop{\mathbb{E}}_{(s_i, s_j)\sim\mathcal{D}}\left[\left(\phi_{\boldsymbol{\theta_R}}(s_i)^\top \phi_{\boldsymbol{\theta_R}}(s_j)\right)^2 - \|\phi_{\boldsymbol{\theta_R}}(s_i)\|_2^2 - \|\phi_{\boldsymbol{\theta_R}}(s_j)\|_2^2\right],\end{aligned} \tag{2.10}$$

where $T \geq \bar{k} \geq 1$ is sampled from the distribution $p(X = \bar{k}) = \frac{\gamma_L^{\bar{k}-1}}{\sum_{t=1}^{T}\gamma_L^{t-1}}$ and $\gamma_L \in [0, 1]$.

2.4 Generalization in Deep Reinforcement Learning

The problem of generalization in deep reinforcement learning is concerned with learning features that help us learn faster when facing new situations, that is, unseen observations [cf. 32]. These new situations have been instantiated in the field in a multitude of ways; ranging from simply knowing how to behave when visiting a novel state of the underlying MDP, to non-stationary settings in which the underlying dynamics of the problem change (e.g., transitions, rewards). In this latter case, it is often assumed that regularities or invariances are preserved, making generalization possible. One controlled way to assess generalization capabilities is to explicitly design training and testing tasks where training tasks are used for training the agent and testing tasks are used for assessing its generalization performance, allowing us evaluate whether the learned representations facilitate future learning [cf. 32]. This approach for assessing generalization performance is similar to the transfer learning problem, where training tasks are referred to as source tasks and the testing tasks are referred to as target tasks [cf. 96]. In this work we investigate such setting, which we call *feature generalization*, or simply generalization from here

onwards. Our goal is to assess when deep reinforcement learning algorithms are able to learn features that facilitate future learning.

Successful generalization in deep reinforcement learning is extremely challenging [e.g., 15, 94], especially when we do not train our agents on the testing tasks, i.e., we do zero-shot transfer. Even in a simple navigation maze setting without any obstacles, small changes in the initial state can have a detrimental impact on the generalization performance of standard deep reinforcement learning algorithms [93]. Generalization is also a major problem when we continue learning on the testing tasks. Agents pre-trained on training tasks cannot often outperform agents trained from scratch when the testing tasks are the same as the training tasks with only minor changes to the visuals of the input states or the transition function [15, 20, 87].

Many studies have shown that as we increase the number and variety of training tasks, the generalization performance tends to improve as well [e.g., 9, 93, 94]. However, the variety of training tasks plays a key role. Once the agent has experienced a sufficient number of different environments, adding more training tasks has no noticeable impact on the generalization performance [93].

The remaining issue is to reduce the number and variety of training tasks required for deep reinforcement learning algorithms to generalize well, i.e., improve their *task-efficiency*. To overcome this issue, many deep reinforcement learning algorithms induce specific inductive biases (i.e., a priori algorithmic preferences) that are shared between training and testing tasks [e.g., 94]. These inductive biases can be in form of architectural changes, data augmentation techniques, or auxiliary losses. For instance, if we know that the variations in color in the environment do not have any effect on the optimal policy, we can use data augmentation techniques to randomize the color of the input images. Preferably, we should look for inductive biases that are shared across many environments. We can design auxiliary losses, for example, to encourage the representation of states to be similar to each other when they are dynamically close to each other [71], or to encourage the optimal policy starting from those states to be similar [1]. We are particularly interested in the impact of auxiliary losses and of activation functions on generalization performance.

Little to no research has been conducted on the representation properties of deep reinforcement learning agents that generalize effectively. To the best of our knowledge, there is only one work that introduces a property, called *effective dimension*, that is able to characterize when and how the representations generalize well [36]. This work, however, focuses on generalization during training over the state space of a single task, whereas our work studies generalization over multiple tasks. This is the first work to provide a generalization framework that allows us to collect and analyze the representation properties discussed throughout the literature, further enabling us to obtain a deeper understanding of the generalization capabilities of deep reinforcement learning approaches.

Chapter 3

Representation Properties

It is generally believed that specific properties are desirable for representations. Initially, these properties inspired researchers to come up with fixed bases representations that are beneficial for a variety of purposes, like improving generalization [e.g., 73]. After the emergence of deep reinforcement learning, several studies followed the same direction and started to encourage the learning process to induce specific properties into the representation [e.g., 86]. Many works, for example, encourage the agent to learn a sparse representation to improve its performance [23, 41, 44, 55, 79, 95]. In this chapter, we take inspiration from these works and introduce a collection of properties to analyze representations and to explain their generalization performance.

We characterize representations into three main axes: *capacity*, *efficiency*, and *update robustness*. Capacity reflects whether a representation can represent a given function. Efficiency captures the lack of redundancy of the features and the computational cost of using them. Update robustness captures the idea that interference is undesirable and that representations should avoid it. We define six properties that capture these three axes, and we use them to evaluate the representations learned by our agents. We design these properties so that measuring them always yields values between 0 and 1. Our goal is to develop a systematic methodology for assessing learned representations based on a diverse set of properties. This evaluation list does not suggest that a property is necessary; rather, it provides some quantitative measures to supplement more qualitative evaluations like visualization. Such a list is

necessarily incomplete; we attempt only to start with a reasonably broad set of properties.

Before diving into the specifics of these properties, we need to present notation that is used to define them throughout this chapter. Building on top of the notation we presented in the previous chapter, the symbol $\boldsymbol{\phi}_i$ refers to the representation of s_i; $\hat{q}_{\boldsymbol{\theta V}}(\boldsymbol{\phi}_i, \cdot)$ is the value network learned given that representation. We compute distances both according to the representation and according to action-values:

$$d_{v,i,j} \stackrel{\text{def}}{=} |\max_a \hat{q}_{\boldsymbol{\theta V}}(\boldsymbol{\phi}_j, a) - \max_a \hat{q}_{\boldsymbol{\theta V}}(\boldsymbol{\phi}_j, a)|, \tag{3.1}$$

$$d_{q,i,j} \stackrel{\text{def}}{=} \max_a |\hat{q}_{\boldsymbol{\theta V}}(\boldsymbol{\phi}_i, a) - \hat{q}_{\boldsymbol{\theta V}}(\boldsymbol{\phi}_j, a)|, \tag{3.2}$$

$$d_{s,i,j} \stackrel{\text{def}}{=} \|\boldsymbol{\phi}_i - \boldsymbol{\phi}_j\|_2. \tag{3.3}$$

The first formula reflects differences in the values the agent uses to select the greedy action; the second looks at the difference in values across all actions; and the last measures the distance between two representations.

3.1 Capacity: Retaining Relevant Information

The first axis to consider for a representation is its capacity: can it represent the functions we want to learn? The value function should be a simple function of these features, such as a simple neural network. To measure capacity, we use one direct measure, *complexity reduction*, and two indirect measures (i.e., without considering the value function), *dynamics-awareness* and *diversity*.

3.1.1 Complexity Reduction

Complexity reduction reflects how much the representation facilitates the simplicity of the learned value function. If the complexity is small, the features encode much of the non-linearity needed to learn a value function. We measure this using the Lipschitz constant of the value network given the representation. The Lipschitz constant measures how fast action values change given the representation. In other words, a higher Lipschitz constant indicates

the value function is more complex to learn. Lipschitz value functions have been motivated for value transfer [42] and learning models [2].

The definition of complexity reduction is shown in Equation 3.4. The Lipschitz constant L is one where $\frac{d_{q,i,j}}{d_{s,i,j}} < L$ for all different pairs of states (s_i, s_j). If the state space is too large or infinite, we can approximate the Lipschitz constant by considering only a subset of states. When this ratio is computed on a given time step t, we use the current action-values. We take a slightly less conservative measure, by averaging across these ratios rather than taking the max. The Lipschitz constant L itself is an imprecise measure of the regularity in the surface: one poorly behaved region could result in a high Lipschitz constant, even if the rest have low local Lipschitz constants. We call these averaged ratios L_{rep}, giving

$$\text{Complexity Reduction} \stackrel{\text{def}}{=} -L_{\text{rep}}, \tag{3.4}$$
$$\text{where} \quad L_{\text{rep}} \stackrel{\text{def}}{=} \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{d_{q,i,j}}{d_{s,i,j}}.$$

We normalize the values of this property between -1 and 0 using $L_{\max}$, computed as the maximum L_{rep} over all representations across all time steps. We, then, subtract the results from 1 to ensure the values reside in the range of $[0, 1]$. We majorly use this normalized version of complexity reduction throughout this thesis as presented below:

$$\text{(Normalized) Complexity Reduction} \stackrel{\text{def}}{=} 1 - \frac{L_{\text{rep}}}{L_{\max}}. \tag{3.5}$$

3.1.2 Dynamics Awareness

We can also indirectly measure capacity by testing whether the representation is **dynamics-aware**: can it retain relevant information to the dynamics of the environment? This means that pairs of states, where one is a successor to the other, should have similar representations, and states further apart in terms of reachability should have dissimilar representations. This measure is in fact related to the Laplacian used for successor features and proto-value functions [48, 70]. Interestingly, a few recent contrastive losses, such as ATC, also strive to induce this property in the representation of deep reinforcement learning agents [66, 71]. We further discuss such methods in Chapter 5.

We measure this property as follows. For every state s_i, we take its successor state s'_i and a random state s_j. If the distance, in representation space, between the successor state is smaller than the distance to a random state, then the representation has high dynamics awareness.

$$\text{Dynamics Awareness} \stackrel{\text{def}}{=} \frac{\sum_i^N ||\boldsymbol{\phi}_i - \boldsymbol{\phi}_{j\sim U(1,N)}|| - \sum_i^N ||\boldsymbol{\phi}_i - \boldsymbol{\phi}'_i||}{\sum_i^N ||\boldsymbol{\phi}_i - \boldsymbol{\phi}_{j\sim U(1,N)}||} \tag{3.6}$$

3.1.3 Diversity

In addition, we can measure the **diversity** of a representation, which is the opposite of *specialization*. If a representation is specialized to one value function, then it likely uses a small subspace of the larger Euclidean space and likely does not produce diverse feature vectors. This specialization may be problematic, as it means the representation is unlikely to perform well when it is transferred to learn another value function. This property is original to this work and has not been discussed in the literature.

To define diversity, we use a ratio between state and value differences. Given two states, s_i and s_j, we compare the distance between their representations ($d_{s,i,j}$) and the distance between their action values ($d_{v,i,j}$). If the value distance is high (i.e., the two state values are very different), then the representation distance is also likely to be high to allow this. The interesting case is when the value distance is low. The representation distance can be high or low, and still allow two states to have similar values, because we project from a higher-dimensional feature vector to a scalar value. A representation with high diversity would have high representation distance when possible, allowing two states to be distinguished even when they have similar values. A representation with low diversity would simply map these two states with similar values to similar representations, specializing to this value function. The diversity metric is defined as

$$\text{Diversity} \stackrel{\text{def}}{=} 1 - \frac{1}{N^2} \sum_{i,j}^{N} \min\left(\frac{d_{v,i,j} / \max_{i,j} d_{v,i,j}}{d_{s,i,j} / \max_{i,j} d_{s,i,j} + 10^{-2}}, 1 \right). \tag{3.7}$$

We normalize it by the maximum distances, to be invariant to value and

representation scales. Diversity can be seen as $1 - \textit{specialization}$. The specialization is lower when $d_{v,i,j}$ is small and $d_{s,i,j}$ is large, causing this ratio to be closer to zero. The specialization is higher when the ratio between $d_{v,i,j}$ and $d_{s,i,j}$ is nearly one. Diversity allows us to indirectly measure capacity, as we can check the level of specialization for a given function without needing to have access to the larger set of possible functions.

3.2 Efficiency: Feature Redundancy

We also study the efficiency of representations showing how easy they make it to learn an arbitrary function. Efficiency can be achieved by reducing redundancy in the representation, i.e., finding linearly independent features. We consider two properties for this axis: *orthogonality* and *sparsity*.

3.2.1 Orthogonality

High **orthogonality** results in linearly independent features, and additionally provides distributed features as well as minimal interference. For example, a technique called factor analysis discovers a dense set of orthogonal (latent) factors to explain the data [62]. This representation is highly distributed, as each feature is used to describe many different inputs. At the same time, interference is reduced: the interference for two states with orthogonal feature vectors is zero under linear updating. As before, we normalize magnitudes and ensure higher orthogonality means that more feature vectors, $\boldsymbol{\phi}_i$ and $\boldsymbol{\phi}_j$, are orthogonal to each other.

$$\text{Orthogonality} \stackrel{\text{def}}{=} 1 - \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{|\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle|}{\|\boldsymbol{\phi}_i\|_2 \|\boldsymbol{\phi}_j\|_2} \tag{3.8}$$

It is worth noting that there is an equivalence between orthogonal feature vectors (i.e., orthogonal representations) and orthogonal features: the sum over all states i, j of $\langle \boldsymbol{\phi}_i, \boldsymbol{\phi}_j \rangle^2$ is equal to the sum over all pairs of features of the dot product between the vector of those feature values across states [cf. 81].

3.2.2 Sparsity

Another property that is connected to efficiency is **sparsity**. If only a small number of features are active (i.e., have non-zero values) for an input, then the features are sparse—with typically the additional condition that each feature is active for some inputs (no dead features). Sparsity is tightly related to orthogonality. For non-negative features, maximizing sparsity corresponds to finding orthogonal features: dot products can only be zero when features are non-overlapping for two inputs. Sparsity has the additional benefit of improving efficiency for querying and updating the function, because only a small number of features are active. For the same reason, sparse representations also reduce interference [44, 55]. To measure sparsity, we calculate the percentage of inactive features on average across a set of states sampled from the environment.

$$\text{Sparsity} \stackrel{\text{def}}{=} \frac{1}{dN} \sum_{i=1}^{N} \sum_{j=1}^{d} \mathbb{1}(\boldsymbol{\phi}_{i,j} = 0), \tag{3.9}$$

where the representation $\boldsymbol{\phi}_i$ for state s_i is d dimensional.

3.3 Update Robustness: Interference Reduction

More recent work in neural networks has also focused on robustness, both to interference and noise. **Interference** reflects how much updates in one state increase error in other states. Given this definition, interference is not desirable because it hurts generalization performance of the agents, especially in the same task [6]. We use a recent measure developed for reinforcement learning [45], which uses the difference in temporal difference errors before and after an update. We do the comparison each time the target network is synchronized, which occurs every k steps, for a total of T times during learning. For every $t = 1, \ldots, T$, we compare the error between $\boldsymbol{\theta}_t$ and the parameters after k updates, $\boldsymbol{\theta}_{t+1}$; note t here references the synchronization iterator rather than time, i.e., $+1$ represents k updates. The result of this comparison for each t is called Update Interference$_t$. We record update interferences up to T and use their average 90th percentile to evaluate interference, which is further used

to compute non-interference.

$$\begin{aligned}
&\text{Non-interference} \stackrel{\text{def}}{=} 1 - \frac{\text{Interference}}{\text{Interference}_{\max}} \\
&\text{Interference} \stackrel{\text{def}}{=} \text{Average}\left(\{\text{Update Interference}_{t \geq Percentile_{0.9}}\}_{t=1}^{T}\right) \\
&\text{Update Interference}_t \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} \text{err}_{t,i}(\boldsymbol{\theta}_{t+1}) - \text{err}_{t,i}(\boldsymbol{\theta}_t) \\
&\text{err}_{t,i}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \left(r_{i+1} + \gamma_t \max_a \hat{q}_{\boldsymbol{\theta}_t}(s_i', a) - \hat{q}_{\boldsymbol{\theta}}(s_i, a_i)\right)^2
\end{aligned} \tag{3.10}$$

Similar to $L_{\max}$ in Equation 3.4, $\text{Interference}_{\max}$ is the maximum interference across all representations.

Chapter 4

Experimental Design

We aim to understand the properties of representations that emerge in deep reinforcement learning in the context of generalization across tasks. To do so, it is critical to study representations that are both good and bad at generalization. More precisely, if representations learned from a collection of training environments allow faster learning on the testing environments, then such representations are considered good at generalization, i.e., they successfully generalize. Keeping this in mind, we carefully design our experiments to ensure that we have a diverse set of representations.

The overall experimental design is shown in Figure 4.1. We conduct all our experiments in a simple maze, where training and testing environments only differ in their goal location. Our experiments consist of two stages: a training stage for learning the representation, and a testing stage for measuring the generalization performance of the learned representation.

The first stage of our experimental design consists of training the representation network. All representations are trained with a DQN agent for $300,000$ steps in the training task. To prevent overfitting, we employ an *early-saving* strategy to save the representation function as soon as the agent is able to successfully finish 100 consecutive episodes in 100 steps or less. Each representation is induced by a choice of activation function and of auxiliary task, including the option of not using an auxiliary task.

In the second stage, we learn with the representation from the first stage, in each testing task. Specifically, we 1) load and freeze the learned representation,

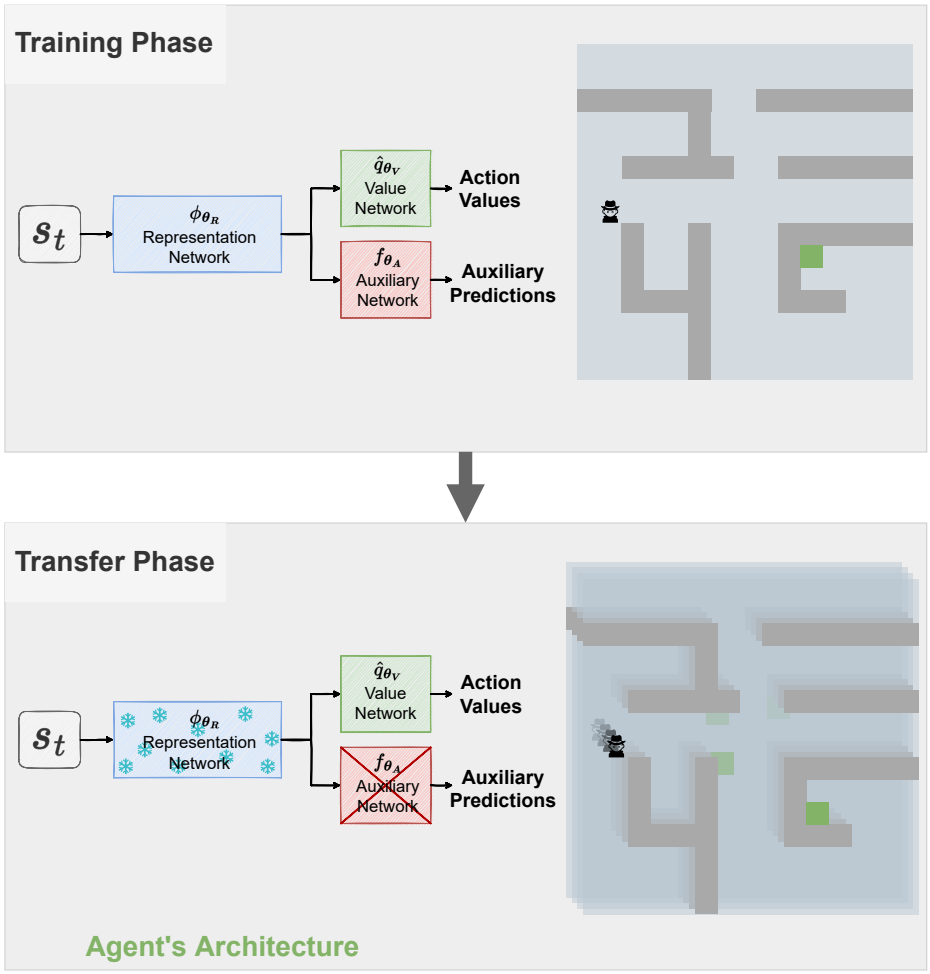

Figure 4.1: Experimental design. In the first stage, all network components participate in the training process. In the second stage, we re-initialize the value network weights, while the representation network is frozen and the auxiliary network is removed. Green denotes the goal location in the environment.

2) re-initialize the value function, and 3) learn the value function for the testing task with DQN, with the fixed representation. No auxiliary tasks are used in this stage, and only the value network is learned with DQN. The agent learns in this new task for $100,000$ steps.

The choices of fixing the representation network and re-initializing the value function are crucial to our experimental design. Freezing the representation function results in representations with fixed properties, allowing us to carefully study the connection between the generalization performance and representation properties. Learning with a re-initialized value function rather than

fine-tuning prevents negative effects from the old value function during testing, especially to less similar testing tasks. Further, re-initializing the value function ensures that the difference between using the pre-trained representations and learning from scratch is due to the learned representation.

We consider 173 testing tasks: all possible goal locations, including the training goal state. To sort performance amongst these tasks, we provide a novel method to measure their similarity to the training task. In this way, we can ask questions about generalization over more or less similar testing tasks.

In the remainder of this chapter, we go over all sub-components of our experimental design as well as our approach for reporting generalization performance and hyperparameters. More specifically, we begin with a detailed description of the maze environment. Then we discuss our agents and their network architecture. Following that, we introduce our approach for measuring the similarity between tasks. After this, we discuss how we report the generalization performance and how we fairly sweep over our agents' hyperparameters. At last, we discuss how we measure representation properties and evaluate the efficacy of different auxiliary tasks.

4.1 Environment

We seek to empirically relate representation properties and generalization performance, which requires an environment where successful generalization is possible. In light of this, we choose a simple pixel-based navigation environment with obstacles, as depicted in Figure 4.1. This environment can be readily used to generate numerous related tasks. The agent must learn to navigate to a given goal state in as few steps as possible. The problem is episodic, with $\gamma = 0.99$, a reward of $+1$ when reaching the goal, and 0 otherwise. The input state consists of an RGB image of a 15×15 grid (size $15 \times 15 \times 3$), encoding the agent's current location (but not the goal). The actions correspond to the four cardinal directions, and transition the agent deterministically by one pixel, or not at all if the action is into a wall. To simplify exploration, the agent starts in a uniform random state and episodes are cut-off at 100 steps; the agent is

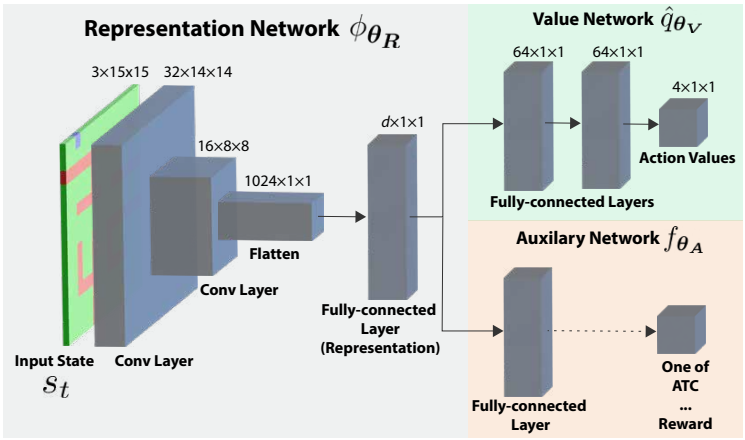

Figure 4.2: The detailed network architecture used in this work. We experiment with agents using this network architecture, with different auxiliary losses. The representation network, $\phi_{\boldsymbol{\theta_R}}$, learns a mapping from input-state s_t to the agent-state (representation of s_t). The representation network is learned to improve two objectives: performance on a main task and on an auxiliary task. Our agents only use one auxiliary task at a time.

then teleported to a new random state and this transition is discarded.

4.2 Agent Specification

We use a unified agent architecture based on convolutional neural networks to explore representations induced by a variety of auxiliary tasks and activation functions. Figure 4.2 depicts this architecture in greater detail. The representation function consists of two convolutional layers, one linear transformation, and a choice of activation function. The linear layer projects the output of the convolutional layer to a 32-dimensional space.

We use two types of activation functions: ReLU and FTA. When using ReLU, the representation layer has $d = 32$ features. If FTA is used, it has 640 features, since FTA projects each scalar to a short, sparse vector with 20 bins. Note that FTA still uses the same number of learned parameters to produce these 640 features as ReLU does to produce the 32 features. This is because binning occurs after the linear weighting in FTA. However, the number of features is higher, and so the value function and auxiliary tasks all

have more parameters, at least in their first layer. We therefore also evaluate ReLU(L)—L for large—which uses 640 features. ReLU(L) uses significantly more parameters than FTA to produce these 640 features.

We use two hidden layers with 64 nodes each for the value function, while the choice of the auxiliary network depends on the complexity of the auxiliary task. We use the auxiliary tasks described in Section 2.3.3. Among these auxiliary tasks, six of them are adopted from the literature and can be applied to any environment with next to no expert knowledge: ATC, IR, NAS, SF, Reward, Laplacian. The remaining three are prediction tasks that are environment-specific and require expert knowledge: VirtualVF-1, VirtualVF-5, XY

We use standard choices for DQN, including the use of ϵ-greedy exploration, an experience replay buffer, target networks, and the Adam optimizer [31]. In total, there are 10 choices for auxiliary tasks (including the choice of not having an auxiliary task) and three activations. When using FTA with auxiliary tasks, we set the number of bins $k = 20$ and $\eta = 0.2$. This implicitly specifies the range for binning to $[-2, 2]$. For the No-aux task agents (i.e., agents without any auxiliary tasks), we test $\eta = 0.2, 0.4, 0.6$, and 0.8, and we report performance for each, not the best one. This gives a total of 33 agent specifications.

We consider three baseline agents, which we call Random, Input, and Scratch. They allow us to falsify different hypotheses about the role of the learned representation. Random uses a randomly initialized network as the representation, without any learning. The agents start with a random network, so this baselines checks whether learning actually improved the representation. Input omits the representation network, and directly inputs the agent's observation to the value function component. It is meant to check if the learned representations play any (useful) role, and if learning from scratch in testing tasks might just have been faster with smaller networks. Finally, Scratch is a DQN agent that starts learning from randomly initialized weights in the testing tasks. The purpose of learning the representation is to learn faster than learning from scratch in the testing tasks. This is the most important baseline,

as it defines whether a learned representation was successful in generalization (i.e., facilitated learning faster than SCRATCH) or not (i.e., was comparable to or worse than SCRATCH).

4.3 Task Similarity

We analyse the similarity between each testing task and the training task to determine how difficult it is to generalize to each testing task. We measure the similarity between two tasks by comparing their successor representations. Formally, the successor representation indicates the expected future state occupancy of state, s', starting from any given state, s, by following the policy π:

$$\psi_{\pi,task_x}(s, s') = \mathbb{E}_{\pi,task_x}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}_{\{S_t=s'\}} \mid S_0 = s\right], \tag{4.1}$$

where $\gamma \in [0, 1)$, $\mathbb{I}$ denotes the indicator function, and $task_x$ represents a task specified by a particular reward and transition function.

We consider the successor representations that emerge under the optimal policy of each task. In this way, successor representations capture both the environment dynamics and the optimal policy of the task in hand. To compute a highly accurate estimate of the optimal policy, π^*, we solve the maze by using a simple tabular algorithm called value iteration that assumes both the reward function, r, and the transition function, p, are given. We consider the state to be the cell number of the agent. This results in a total of $|\mathcal{S}| = 173$ states. The optimal policy, along with the transition function, is further used to compute the successor representation of the task [cf. 46].

To compute the similarity measure, we consider successor representations of all state pairs. This is represented by a set of vectors $\boldsymbol{\psi}_{i,task_x}$, where

$$\boldsymbol{\psi}_{i,task_x} = \left[\psi_{\pi^*,task_x}(s_i, s_1)\ \psi_{\pi^*,task_x}(s_i, s_2)\ \ldots\ \psi_{\pi^*,task_x}\left(s_i, s_{|\mathcal{S}|}\right)\right],$$

and $i \in [1, 2, \ldots, |\mathcal{S}|]$. Each vector $\boldsymbol{\psi}_{i,task_x}$ represents how frequently other states are visited starting from state s_i, i.e., successor representations of state s_i. We define the similarity as the dot product between successor representations of testing and training tasks. We choose the dot product to keep both

angle and magnitude information between concatenated successor representations. The equation for computing this similarity metric is

$$Similarity(task_x, task_y) = \sum_{i=1}^{|\mathcal{S}|} \left\langle \boldsymbol{\psi}_{i,task_x}, \boldsymbol{\psi}_{i,task_y} \right\rangle \tag{4.2}$$

Using this similarity metric, we rank the representations based on their similarity to the training task. This ranking is depicted in Figure 4.3. As clear from this figure, the similarity decreases as the distance between the location of the goal for the testing task and the training task increases. We expect this to happen since the optimal policy of tasks with far-away goal locations has significantly less overlap than tasks with close goal locations (i.e., reward function and terminating state).

4.4 Reporting Performance and Hyperparameters

To report performance, we have to consider how to measure the generalization performance and how to set hyperparameters. In both training and testing tasks, we record, every $10,000$ steps, the average return of the last 100 episodes. We take the sum of these recorded values, also called the Area Under Curve (AUC), to summarize the performance across the 300,000 steps. The AUC is used to select hyperparameters.

The only hyperparameter common across all agents is the step-size; we only sweep this hyperparameter. We pick the step-size independently in the training and testing stage. We use the average performance over 5 runs to select the step-sizes. Specifically, in the training stage, we run each of the 33 agent specifications with different step-sizes for 5 runs. We select the best step-size according to training AUC and use the representations produced under these step-sizes. Then in the testing stage, we evaluate each step-size only for these representations, and pick the best step-size for an agent specification for each testing task by using the average performance across the 5 runs. We sweep the step-size to ensure we are evaluating reasonably well-optimized agents.

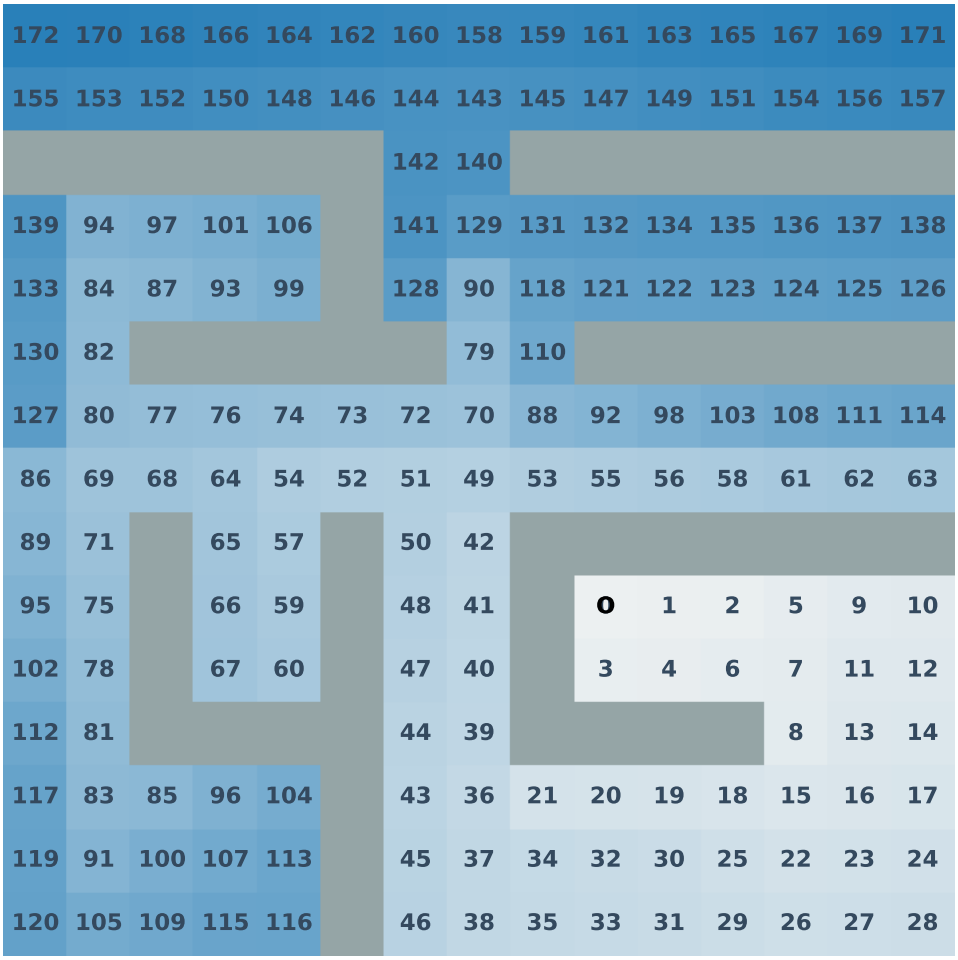

Figure 4.3: Testing tasks raked by their similarity to the training task. This figure shows the similarity ranking of different tasks compared to the training task, where the training task is marked by **O**. Each number in the cell indicates the similarity rank of the task when the goal is moved to that specific position.

Auxiliary tasks also have their own set of hyperparameters that require tuning. However, the only hyperparameter that is shared between auxiliary tasks is the weight of the auxiliary loss, w_{aux}, which specifies how much it contributes to learning the representation. If this weight is not properly set, the agent will fail to solve the training task. To avoid this, we start our experiments with a weight of $w_{aux} = 1$ for each auxiliary task. If the agent fails to solve the task in any of the 5 runs, we decrease the weight to $w \times 0.1$ and repeat the experiments. We repeat this procedure until the agent solves the primary task.

We consider the generalization performance of each agent as its AUC on

each testing task. When we report the generalization performance across agents, we do not average across these 5 runs. Instead, each run produces a different representation and we report performance for each one as an independent data point. When showing aggregate performance, we aggregate from this larger pool of 5 runs for each 33 agents specifications, namely over 165 representations. We do so because each representation has different properties; when correlating agent properties and performance, we may not care which auxiliary task was used, but rather only care about its emergent properties. Averaging across runs compares methods (agent specification), rather than representations.

Finally, we obtain generalization performance in 173 testing tasks. This means we get 173 generalization performance samples for each of the 165 representations. In total, when aggregating across testing tasks or agent specifications, we obtain a significant number of samples to estimate aggregate performance, even though each agent specification only has 5 runs in the training task. In total, we generate $173 \times 165 = 28,545$ agent-task combinations.

4.5 Evaluating the Learned Representations

We use the data generated by these agent-task combinations to evaluate the efficacy of the properties as well as auxiliary tasks and activation functions. We evaluate (measure) the properties using a test set of 1000 transitions, $\mathcal{D}_{\text{test}} = \{(s_1, a_1, s'_1, r_1), (s_2, a_2, s'_2, r_2), \ldots, (s_N, a_N, s'_N, r_N)\}$. This dataset is obtained by running the random policy for $N = 1000$ episodes, and then randomly subsampling N transitions, to ensure we cover the state space. This sampling approach can be easily used for other choice of environments through adjusting the value of N by the complexity of environment. It is worth noting that in many real-world scenarios, the agent has access to a large quantity of offline data [cf. 43], which may be employed instead of this approach to generate the test set. In Chapter 5, we visualize these properties and their respective generalization performance and study which value and combination of these properties result in better generalization performance. We also visual-

ize the generalization performance of the auxiliary tasks and activations across testing tasks that are ranked by their similarity to the training task. Doing so enables us to identify which auxiliary tasks help the agent in performing consistently across both similar and dissimilar tasks. These agent-task combinations further enables us to investigate which components of the auxiliary tasks are most essential for improving the agent's generalization performance across tasks.

It would be difficult to guarantee that the conclusions of our study generalize beyond the specific generalization setting investigated here. But these properties and our methodology can be used to understand representations in other generalization settings in a systematic and reproducible way toward the ultimate goal of understanding good representations for reinforcement learning.

Chapter 5

Empirical Investigation of Representation Properties and Generalization Performance

We start this chapter by presenting two results that are crucial to our empirical study. First, *successful generalization is possible in our navigation environment and auxiliary tasks improve generalization.*. Figure 5.1 summarizes these results. Representations pre-trained on the training task outperformed representations learned from scratch on the testing tasks. In addition, we found auxiliary tasks were important for generalization, at least for ReLU. Pre-trained representations using ReLU did not generalize well, whereas ReLU-based representations combined with well-designed auxiliary tasks successfully generalized. To the best of our knowledge, no prior work has demonstrated that auxiliary tasks improve representation transfer in reinforcement learning (i.e., generalization in a setting where only the representation function transferred to the testing task); instead most works with auxiliary tasks use them during learning [8, 21, 28] or on an offline dataset [26, 90].

Second, *we were unable to obtain successful generalization using ReLU activation function with linear value functions.* This outcome was not for a lack of effort. Figure 5.1 strongly suggests that, in our navigation environment, non-linear value functions significantly improve transfer, even in a relatively simple environment. We see that when the value function was linear in the features and ReLU activation function was used, neither a pre-trained repre-

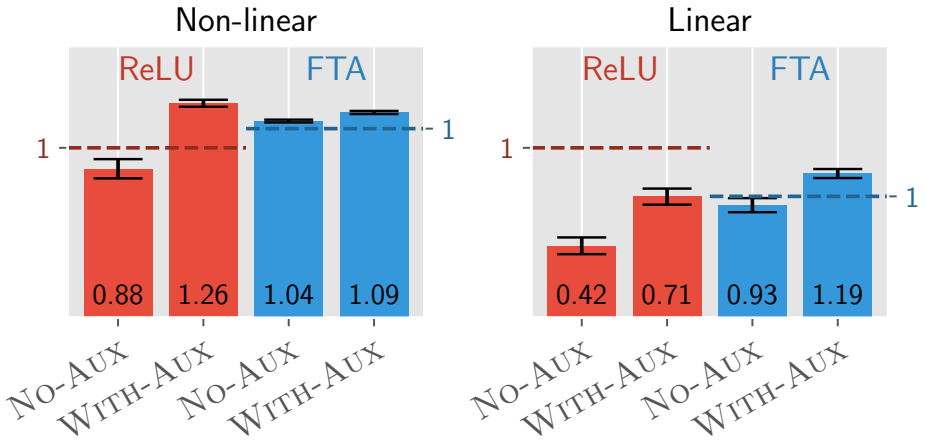

Figure 5.1: Generalization performance of our best representations when using non-linear and linear function approximation. Generalization is possible in the Maze navigation task and auxiliary tasks improve generalization performance when using non-linear function approximation. The performance relative to the baseline (Scratch) agent: above 1 represents successful generalization and below 1 denotes failed generalization. Error bars show a 95% confidence interval. Performance is reported for the best auxiliary task for each activation: VirtualVF5 for ReLU-based representations, and SF for FTA. The values shown on the bars indicate the relative performance to FTA-based or ReLU-based Scratch. However, the location of the dotted line and the length of the bars represent overall generalization performance. In the linear case, the overall generalization performance of ReLU is 8.29 and FTA is 5.92, and in the non-linear case, this performance for ReLU is 7.63 and for FTA is 8.49. This performance is calculated over 5 runs of 173 testings tasks: $173 \times 5 = 865$ samples. In non-linear, FTA baseline is better; in linear, ReLU baseline is better.

sentation (labelled ReLU) nor any other representations trained with auxiliary tasks improved over training a fresh representation from scratch. This suggests representations may emerge in earlier layers of the network, and that it may be more feasible to learn re-usable features when they can be nonlinearly combined, even if only with a simple shallow network. We highlight this important point here; for the remainder of this thesis, we restrict our scope to non-linear value functions.

In the following sections, we first investigate how the generalization performance of the representation is affected by combination of different activation functions and auxiliary tasks. At last, we investigate the properties of these

representations and shed light on the main thesis of this work. The hyperparameters used for all of our experiments are detailed in Appendix A.

5.1 Generalization Performance of Good, Bad, and Ugly Representations

We expect some agent specifications to result in representations that aid generalization performance and others to impede it. UNREAL [28], the first large-scale deep reinforcement learning system to highlight the utility of auxiliary tasks, showed that although auxiliary tasks like pixel prediction improved performance substantially, other tasks such as feature control had a much smaller impact. Other work has highlighted that it can be difficult to achieve good generalization in reinforcement learning [e.g., 15, 78]. It seems the design and deployment of auxiliary tasks remains largely an art.

In this section, we provide some clarity on these discrepancies by showing that 1) there is large variability in performance across auxiliary tasks, and 2) generalization performance can degrade significantly as tasks become more dissimilar. Though these results are intuitive, they constitute the first approach to systematically vary these two axes to understand when methods may be succeeding or failing.

5.1.1 Generalization Performance on Similar and Dissimilar Tasks

Figure 5.2 summarizes the generalization performance of many different representations corresponding to different auxiliary tasks and activation functions. The plot has task similarity on the x-axis, and each point on the plot summarizes the performance of one representation on one particular testing task. The lines show how much generalization performance degrades as tasks become more dissimilar.

The bold black line shows performance in the testing tasks if the representation and value function were trained from scratch. Any point above the bold black line indicates a representation that achieved better generalization per-

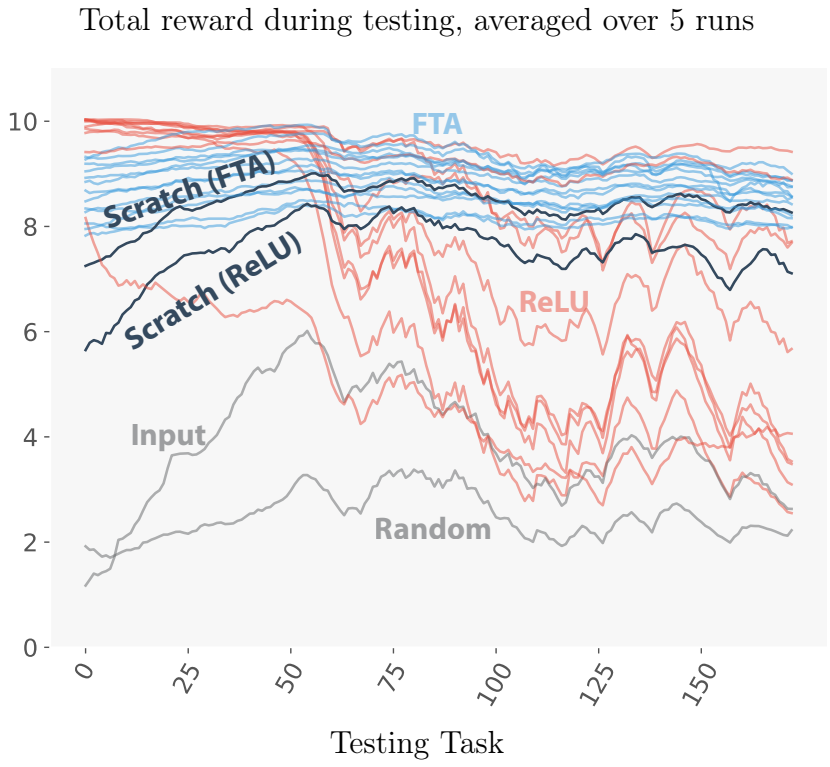

Figure 5.2: Generalization performance of 115 different representations (ReLU and FTA) on 173 testing tasks. The tasks on the x-axis are arranged by similarity to the training task: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. The black line shows the performance when learning in each transfer task from scratch. Lines completely above the black line indicate a representation yielded successful generalization in all tasks. Lines that start above the black line but fall below as we move left to right indicate a representation that generalize well to similar tasks but not dissimilar tasks. The INPUT and RANDOM baselines are not competitive; for completeness, we still report their performance, but with lighter lines. Overall, many representations successfully generalize and generally FTA-based representation are better on this problem. Details on how task similarity was computed and how this plot was generated can be found in Section 4.3. The generalization performance of ReLU(L) is compared with ReLU in Figure B.2 in Appendix; it exhibits the same pattern as ReLU, generalizing well to similar tasks and not as well in less similar tasks.

formance than training from scratch on that task. Any line completely above its corresponding black line indicates a representation that achieved successful generalization for all goal states.

The most important conclusions from Figure 5.2 are that (1) several representations achieve successful generalization across all tasks, and (2) a great va-

riety of representations emerge with generalization performance ranging from good to significantly worse than Scratch. Looking more closely, some representation achieve successful generalization in dozens of tasks which are most similar to the training tasks, but for tasks very dissimilar to the training task, performance is poor, as seen by the step down in many of the lines.

Generally, we found that FTA-based representations yield better representations (in terms of generalization performance) compared to ReLU. Almost all FTA representations outperformed Scratch (FTA), and generalizing to less similar tasks was effectively the same as generalizing to similar tasks, as evidenced by the nearly flat lines in Figure 5.2. Interestingly, many ReLU-based representations did extremely well in similar tasks but significantly worse than Scratch (ReLU) (nearly as bad as the input baseline) in less similar tasks. ReLU(L) performed similarly to ReLU, with ReLU(L) performing slightly worse on similar tasks and slightly better on dissimilar ones (this result is in Appendix B.1.1). Another point of interest is that Scratch (FTA) performs better than Scratch (ReLU), and yet FTA representations generalize better than ReLU-based ones: using FTA improved on using ReLU and then training the FTA representation first in a training task improved performance in the testing task even more.

5.1.2 Generalization Performance of Individual Auxiliary Tasks

Digging a little deeper, Figure 5.3 depicts the generalization performance of each auxiliary task. We again see that FTA-based representations achieve higher performance overall and higher performance across auxiliary tasks (the worst performing representation never used FTA, always ReLU). Inspecting each auxiliary task, the FTA-based representations exhibited lower variance across runs. Larger ReLU representations, ReLU(L), did improve performance over the smaller ReLU representations, but not uniformly. The IR auxiliary task representation, for example, improves with large ReLU networks, but ATC performs worse (though not significantly in either case).

At the auxiliary task level, there are no obvious trends (except for the

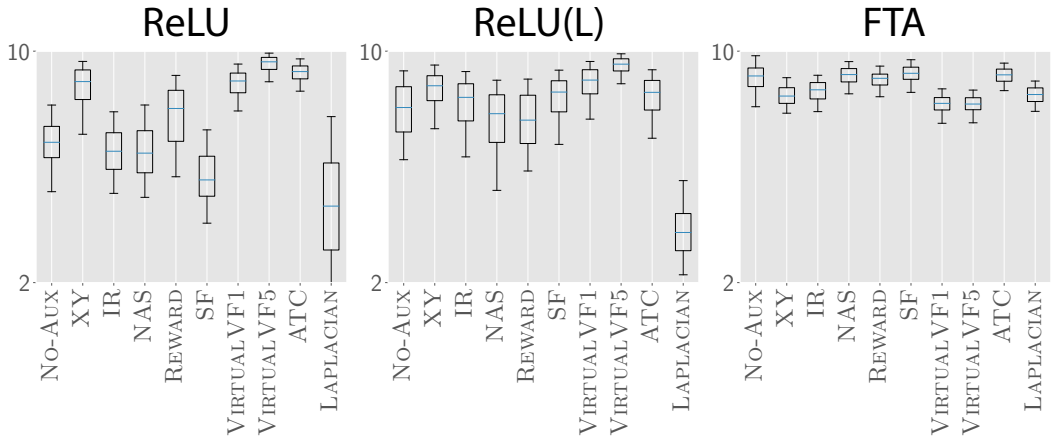

Figure 5.3: Generalization performance based on the activation function, representation size, and auxiliary task. Overall, FTA-based representations achieved the best performance and exhibited the least variation in performance across auxiliary tasks. The orange lines depict the median, the upper and lower edges of the box show the 25_{th} and the 75_{th} percentiles, while the whiskers show 1.5 times the inter-quartile range. These results are computed over $173 \times 5 = 865$ samples, and so the standard errors are quite small (as you can see in Figure B.4 in Appendix B.1.3).

fact IR, Reward, and NAS are generally not useful). For example, SF is among the best performing FTA and ReLU(L) representations and among the worst performing ReLU representations. The subgoal-navigation auxiliary tasks (VirtualVF) result in the best performance with ReLU representations. These subgoals can be thought of as way-points placed at strategic locations in the environment; perhaps these tasks force the network to represent how to navigate to these waypoints which then speeds learning when navigating to other nearby goals in the testing stage. Perplexingly, these are not the best performing representations when combined with FTA activation functions. Perhaps FTA networks already extract a general and transferable representation (as evidenced by the performance of No-Aux), and thus the subgoal auxiliary tasks simply do not help much. It is difficult to know looking at performance only; in the following sections we look at different properties of the learned representations as a lens to understand such mysteries.

In the case of ReLU, ATC, along with VirtualVF-5, is the only loss that achieves high generalization performance in all testing tasks. These results are

presented in Figure B.3 in Appendix B. ATC is also the only loss that achieves better or similar performance to No-Aux representations across all activation functions. Despite these compelling results, we realized that omitting data augmentation from ATC loss significantly degrades its performance with ReLU representations. At the same time, we found that the ReLU network combined with data augmentation (random shifting of input images) and no contrastive setup achieved similar performance to ReLU ATC. This is shown in Figure B.5 in Appendix B.

The capacity of the neural network (i.e., number of nodes in the representation layer) plays an important role in determining the generalization performance of auxiliary tasks. SF is one of the worst ReLU-based representations. However, when we increase the size of the representation layer (using ReLU(L)), it achieves similar performance to the representation without auxiliary task (No-Aux representation). Reward task experiences the opposite. We speculate that this is because predicting the next state reward is a simple task and small networks are able to capture it. In this case, increasing the size of the representation layer leads to overfitting to the task at hand. However, predicting the successor features is a complex task that cannot be captured using a small representation layer. In this instance, SF's updates will mostly function as noise rather than as an instructive signal that might improve the representation.

We included Laplacian to see what will happen if we directly encourage the representation to be dynamics-aware. Laplacian worked extremely poorly, by being the worst ReLU and ReLU(L) representation. Interestingly, we found that decreasing the value of γ_L to values such as 0.25 and 0 greatly increases its performance. However, low values of γ_L do not help the representation in capturing the eigenvalues of the Laplacian matrix. We discuss this in more detail in Chapter 6.

5.2 The Properties of Good Representations

We can now return to the main question of this work: do good representations that generalize well exhibit particular properties? In this section, we first investigate how the properties defined in Chapter 3 relate to generalization performance. Then, we investigate how representation properties change over the training process and whether they converge. Lastly, we study the properties of certain representations to explain why they do or do not generalize well.

Figure 5.4 contains the main result; for now let us focus only on the FTA representations in the top row to better understand this figure. Each subplot shows the generalization performance of every single FTA-representation averaged over all testing tasks. The representations in each subplot are ordered based on a single measured property. For example, the first subplot in the first row plots the generalization performance of FTA representations, and the dots are ordered by complexity reduction on the x-axis. Representations with high complexity reduction and good generalization performance would appear as a dot in the top right of the subplot. Representations with low complexity reduction and good generalization performance would appear as a dot in the top left of the subplot, and so on.

At the highest level we see FTA (first and third row) and ReLU (second and third row) exhibit different properties across representations. FTA-based representations, by large, exhibit high complexity reduction and high diversity, whereas ReLU representations range widely from low to medium on the same two measures. In fact, the *lowest* observed complexity reduction and high diversity of any FTA representation was greater than the *highest* observed complexity reduction and high diversity for ReLU. ReLU representations could be sparse and have low or high orthogonality, whereas FTA representations are mostly sparse. Interestingly, the top representations in terms of sparsity were ReLU. ReLU representations with similar property values can achieve very different generalization performance (visible as points stacked vertically). There appears to be no clear relationship between sparsity and performance

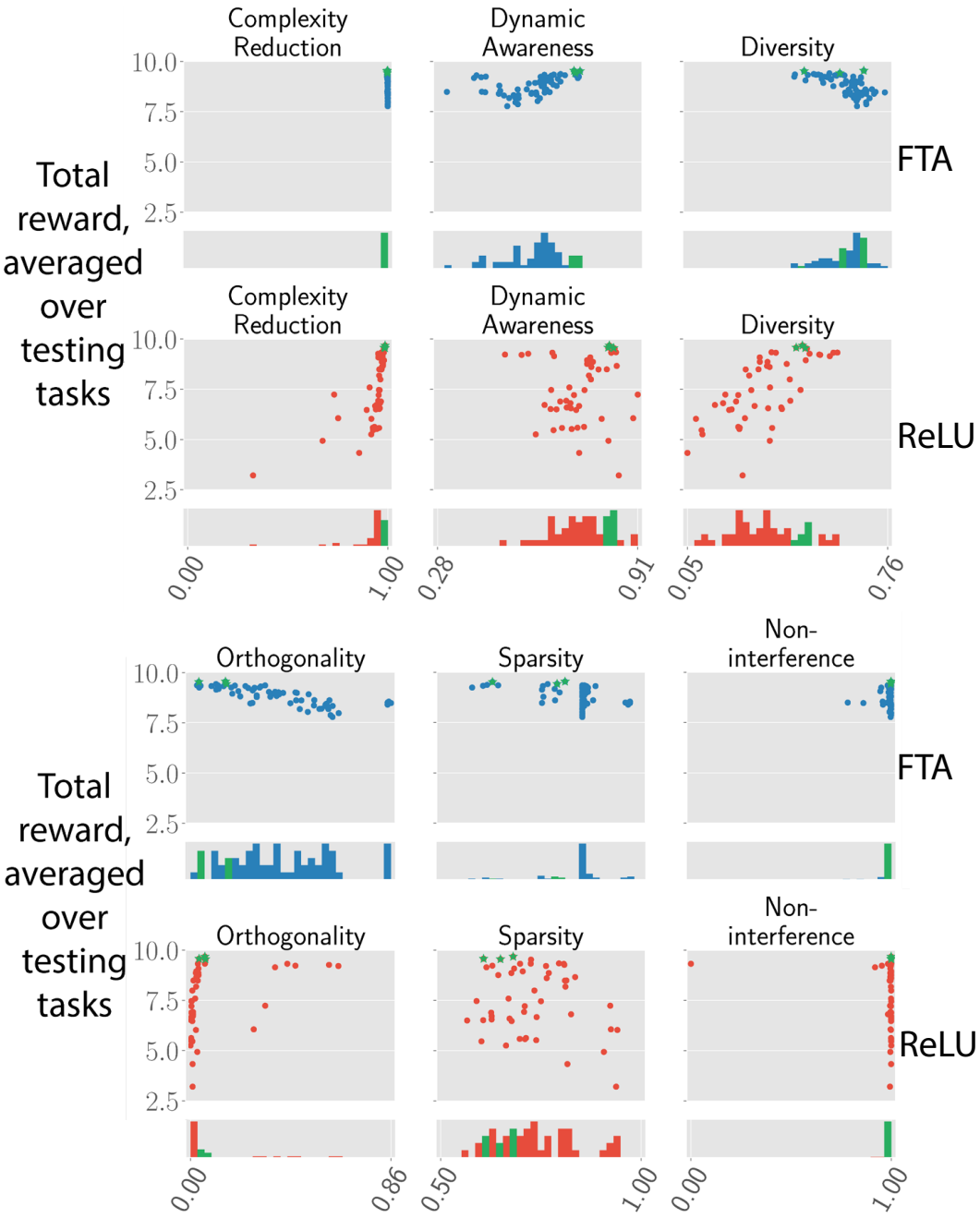

Figure 5.4: Performance averaged over testing tasks versus representation property values. Each dot in a plot corresponds to one representation, at the (x,y) point corresponding to its property value and average generalization performance. The histograms under the plots visualize the density of points at each property value. The green stars correspond to the three best performing representations. We separate out FTA-based and ReLU-based representations, which exhibit notably different behavior.

for ReLU representations.

Now consider the properties of the top performing representations. Again, let us focus our attention on the FTA representations in the odd rows of Figure 5.4. The green stars in each subplot correspond to the top performing representations (in terms of generalization performance). First notice the stars are typically close together in x and y indicating all three achieve similar performance with similar property values; this is true for ReLU representations as well. In general, we see that the best performing representations are not at the extremes of any property (high or low). Given that FTA representations by large exhibit high complexity reduction, diversity and sparsity, it is notable that the best performing representations are the lowest of those three properties.

In general, the best representations for both FTA and ReLU exhibit fairly similar properties (relative to other representations with the same activation function): high complexity reduction, high dynamics awareness, high diversity, and low orthogonality. Of particular note is the clear pattern in complexity reduction and diversity for ReLU: both needed to be higher, and performance clearly drops for lower values. FTA seems to more naturally produce representations that are higher on these measures; we hypothesize that this is the main explanation for why FTA representations perform well across all testing tasks.

It is true that our best representations have low orthogonality. However, it is important to note that as orthogonality approaches 0 (looking at the ReLU representations), performance suddenly degrades. In other words, the representations in the first bin of the orthogonality histogram perform poorly. In addition, a handful of representations with high values of orthogonality are still able to achieve reasonable generalization performance. This might indicate that representations benefit from some degree of orthogonality, but the absence of it is detrimental.

The reverse is true for complexity reduction: we see a pattern of reduction in performance as complexity reduction approaches its maximum value. To clearly show this, we plot the unnormalized version of the complexity reduction (presented in Equation 3.4) in Figure B.6 in Appendix B. The performance of

the representations improves as the complexity reduction approaches the value of -0.03. After this point, however, performance gradually decreases. This makes sense because complexity reduction reflects how easy it is to learn the value function for the training task. Therefore, if learning the value function with the representation is overly simple, the representation is overfitted to the value function of the training task.

Non-interference for all the representations is high. This makes it hard to make any judgment about the connection between non-interference and generalization performance. We think the reason for this is that the maze environment has a low number of states (173 states) and the agent has a huge replay buffer, making it easy for the agent to memorize the value of all states. One way to overcome this issue is to extend this study to more complex environments, where the number of states is much higher.

5.2.1 On the Convergence of Representation Properties

The property values depicted in Figure 5.4 were computed from the representations when frozen for testing, but one might wonder what are the dynamics of the properties over time. Recall that we froze and transferred each representation to the testing phase after 100 consecutive episodes were successfully completed in the training phase. This choice balances the need for reasonable performance without having to select somewhat arbitrary budgets on the number of steps or performance criteria. However, our choice does mean that each representation could receive different amounts of experience, and thus begs the natural question: would the properties reported in Figure 5.4 be very different with more or less training. Figure 5.5 provides the answers.

Generally, across all auxiliary tasks and activation functions, the representation property values remained similar after initial transients in early learning. Each subplot in Figure 5.5 shows a particular property value for every single representation tested over an extended training period. We intentionally do not distinguish between activation functions and auxiliary tasks in this plot. The change in color indicates when the representation was frozen for testing, in terms of training time. Note that many representations were frozen af-

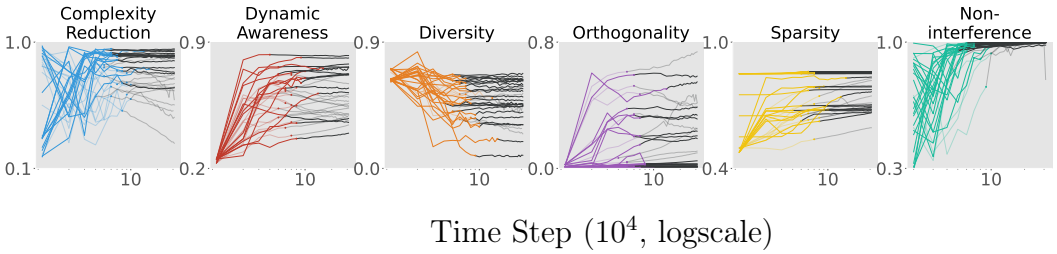

Figure 5.5: Plotting the representation properties over time, with one subplot per property. Each curve shows the property of one agent specification (activation and auxiliary task pair), averaged over the 5 runs in the training task. The curve changes color, to black, at the time point where we took the representation and fixed it; this point was chosen based on when the return for the agent stopped changing. Line colors vary from light gray to black based on how much their value fluctuate from the property value after their return converges, with darker lines denoting lower variation. In these plots, we allowed the representation to keep learning to understand if properties significantly change afterwards. Our primary focus is to show the general trend that properties converge over time, and that they converge approximately when the return does; therefore we use the same color for all agent specifications.

ter the same number of training steps. Orthogonality, dynamics awareness, and sparsity of a small number of representations slowly increase with more training, and complexity reduction of a few representations slowly decreases. Overall, the properties for the most part converge, and do so just before the representations were frozen for testing. Training the representations for longer would not have resulted in significant changes in the property values.

5.2.2 Explaining Why Some Representations Generalize Well and Some Do Not: An Example

In this section, we use the representation properties to investigate and explain why some representations generalize well and some do not. We particularly focus on VirtualVF5 and investigate why it was helpful for ReLU-based representations and harmful for FTA-based representations. We plotted properties for just these representations in Figure 5.6, and found that the addition of this auxiliary loss significantly decreased dynamics awareness for FTA—to the detriment of performance—but increased it for ReLU. Additionally, it caused the FTA-based representation to have much higher orthogonality, likely

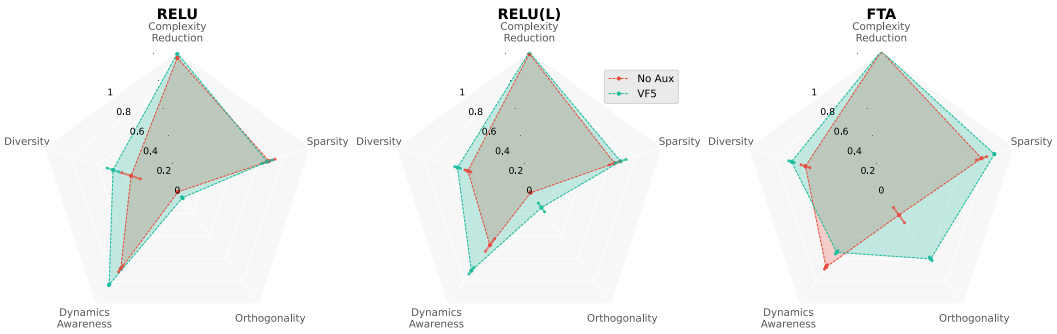

Figure 5.6: Properties of VirtualVF5 representations. The VirtualVF5 task produces FTA representations that do not generalize well, but ReLU and ReLU(L) representations that do. Each subplot shows a property value achieved by four different representations: ReLU, ReLU(L), and FTA with VirtualVF5, and ReLU, ReLU(L), and FTA with no auxiliary tasks. It is clear this auxiliary task changes the properties of the representations; particularly dynamics awareness and orthogonality. We did not include non-interference as VirtualVF5 had no impact on it.

increasing it to one of the extremes that performed poorly. For ReLU, the increase in orthogonality was to an interim level, from a very small value. It is as yet unclear why this auxiliary loss caused these effects, but this clear and systematic change in properties helps explain this outcome.

We also use this approach to explain why the Laplacian representation performed so poorly and why removing data augmentation from the ATC task damaged its performance. These results are presented in Figures B.8 and B.7 in Appendix B, discussing the importance of complexity reduction, diversity, and dynamics awareness.

Chapter 6

Property-based Auxiliary Tasks

So far, we have focused on using auxiliary losses and activation functions to induce a diverse set of properties in the representations. Here, we take a different approach by directly controlling the maximization of desirable properties in the representation. Specifically, we focus on encouraging representations to maximize two pairs of properties: **complexity reduction and orthogonality (CR+O)**, and **dynamics awareness and orthogonality (DA+O)**. The finding that ReLU and ReLU(L) representations exhibit orthogonality around 0 provides the motivation to maximize this property. In the previous chapter, we also showed that high dynamics awareness and complexity reduction help to improve generalization performance. In this chapter, we introduce these two losses and study their generalization performance and properties.

Throughout this chapter, we consider the ATC loss to be representative of our best representations. This is because ATC is the only existing loss in the literature that achieved high generalization performance in all testing tasks with ReLU representations. The hyperparameters used for these property-based auxiliary losses are given in Appendix A.

6.1 Dynamics Awareness & Orthogonality

We use the following auxiliary loss function to maximize dynamics awareness and orthogonality:

$$\mathcal{L}_{\text{DA+O}}(\mathcal{D}) = \mathop{\mathbb{E}}_{(s_t, s_{t+1}) \sim \mathcal{D}} \left[\|\phi_{\boldsymbol{\theta_R}}(s_t) - \phi_{\boldsymbol{\theta_R}}(s_{t+1})\|_2^2 \right]$$
$$+\beta_{\text{DA+O}} \mathop{\mathbb{E}}_{(s_i, s_j) \sim \mathcal{D}} \left[\left(\phi_{\boldsymbol{\theta_R}}(s_i)^\top \phi_{\boldsymbol{\theta_R}}(s_j) \right)^2 - \|\phi_{\boldsymbol{\theta_R}}(s_i)\|_2^2 - \|\phi_{\boldsymbol{\theta_R}}(s_j)\|_2^2 \right]. \tag{6.1}$$

In this equation, both expectations participate in maximizing dynamics awareness. Recall that for dynamics awareness, we want representations of the states that reside in each other's neighborhood to be similar. The first expectation exactly does this by minimizing the distance between representations of subsequent states. We also want the representations of the remaining states to be different, which is encouraged by the second expectation. The second expectation encourages orthogonality as well. This is achieved by encouraging the representations' norms to be high and the representations to be orthogonal to one another. Interestingly, this loss is similar to the LAPLACIAN loss, the only difference being the use of a fixed value of $\bar{k} = 1$ (referring to Equation 2.10). Interestingly, this loss is similar to the LAPLACIAN loss, the only difference being the use of a fixed value of $\bar{k} = 1$ (referring to Equation 2.10).

The generalization performance of DA+O representation is presented in Figure 6.1. As shown, adding DA+O as an auxiliary loss for any kind of activation improves the performance over SCRATCH, except for the case of ReLU(L). This loss also improves generalization performance on all NO-AUX representations (i.e., ReLU, ReLU(L), FTA). This improvement is significant in the case of ReLU in such a way that the performance becomes stable across all testing tasks, reaching a similar performance to our best baseline (ATC). These results show that encouraging representations to be dynamics-aware and orthogonal is a promising approach to improve generalization performance.

We compare the properties of DA+O representations across three activation functions in Figure 6.2. The first thing we notice is that this loss significantly increases dynamics awareness, orthogonality, and sparsity in all cases. Surprisingly, diversity and complexity reduction are also improved in all cases, except in the case of ReLU(L). This is sensible because DA+O combined with ReLU(L) had a hard time surpassing SCRATCH baseline, showing that it could not generalize successfully. These results confirm our previous

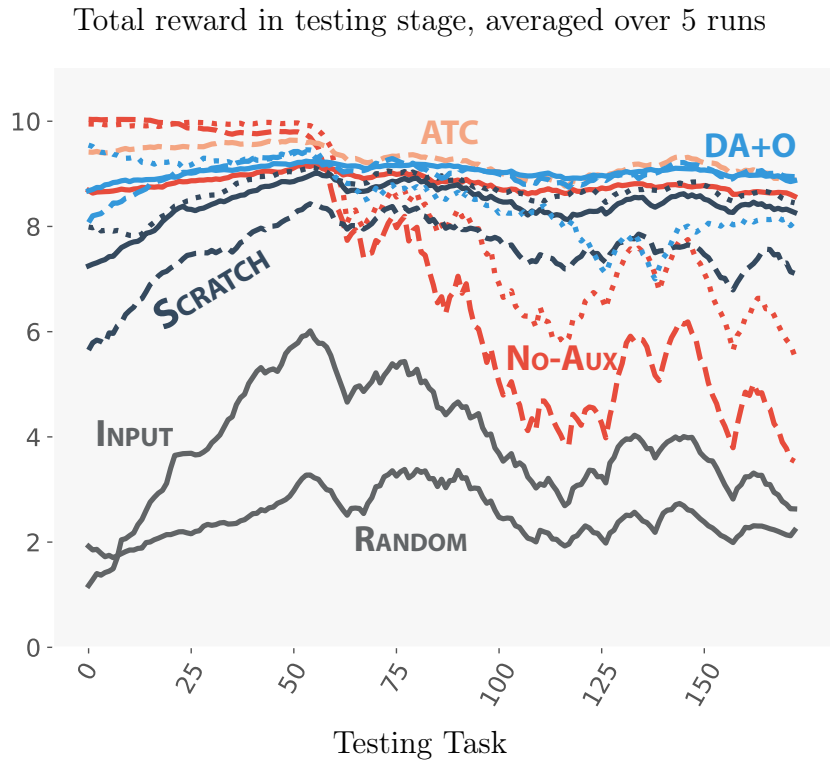

Figure 6.1: Comparing generalization performance of (DA+O) representations with the relevant baselines. The tasks on the x-axis are arranged by similarity to the training task: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. Dashed lines, dotted lines, and solid lines represent ReLU, ReLU(L), and FTA representations respectively. Adding (DA+O) to all three activations leads to improvement in generalization performance.

claim that representations with high dynamics awareness, diversity, complexity reduction, and an interim level of orthogonality achieve good performance. Reduction in one of these properties can hinder performance, as we saw in the case of ReLU(L).

6.2 Complexity Reduction & Orthogonality

We directly use the unnormalized complexity reduction property given in Equation 3.4 to design this loss function. To encourage representations to have high complexity reduction, we only need to maximize this value. This can be easily achieved by using the negative value of this complexity reduction measure as part of our loss. We also add an $\varepsilon_{\text{CR+O}} = 1e-8$ to the denom-

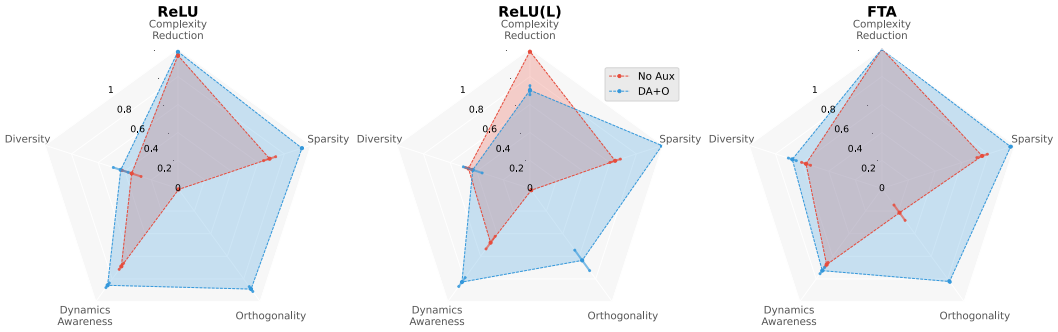

Figure 6.2: Properties of DA+O representations. DA+O representations generalize well across all activation functions. Each subplot shows five property values achieved by six different representations: ReLU, ReLU(L), or FTA with and without DA+O. This auxiliary task significantly changes sparsity, dynamics awareness, and orthogonality. We did not include non-interference as DA+O had no impact on it.

inator of this measure to make sure the denominator is always nonzero. To maximize orthogonality, we use the same expectation as we used for dynamics awareness. Taking this approach, CR+O loss is defined as

$$\begin{aligned}\mathcal{L}_{\text{CR+O}}(\mathcal{D}) = &\mathop{\mathbb{E}}_{(s_i,s_j)\sim\mathcal{D}}\left[\frac{d_{q,i,j}}{d_{s,i,j}+\varepsilon_{\text{CR+O}}}\right]\\ +\beta_{\text{CR+O}}&\mathop{\mathbb{E}}_{(s_i,s_j)\sim\mathcal{D}}\left[\left(\phi_{\boldsymbol{\theta_R}}(s_i)^\top\phi_{\boldsymbol{\theta_R}}(s_j)\right)^2-\|\phi_{\boldsymbol{\theta_R}}(s_i)\|_2^2-\|\phi_{\boldsymbol{\theta_R}}(s_j)\|_2^2\right].\end{aligned} \tag{6.2}$$

The generalization performance of CR+O representation is presented in Figure 6.3. This loss improves performance over No-Aux only when combined with ReLU. In other cases, performance degrades significantly. The representation collapses in the case of ReLU(L), resulting in a performance worse than Random representation. This is surprising because, while this loss significantly improves the performance of ReLU, it destroys the generalization performance of ReLU(L) and FTA. This may indicate that this loss is susceptible to overfitting when the number of features in the representation layer is high.

The properties emerged from maximizing this loss are shown in the Figure 6.4. This loss successfully increases the orthogonality and complexity reduction (complexity reduction is fully maximized in all cases). In the case of FTA and ReLU(L), this loss pushed the value of orthogonality and sparsity

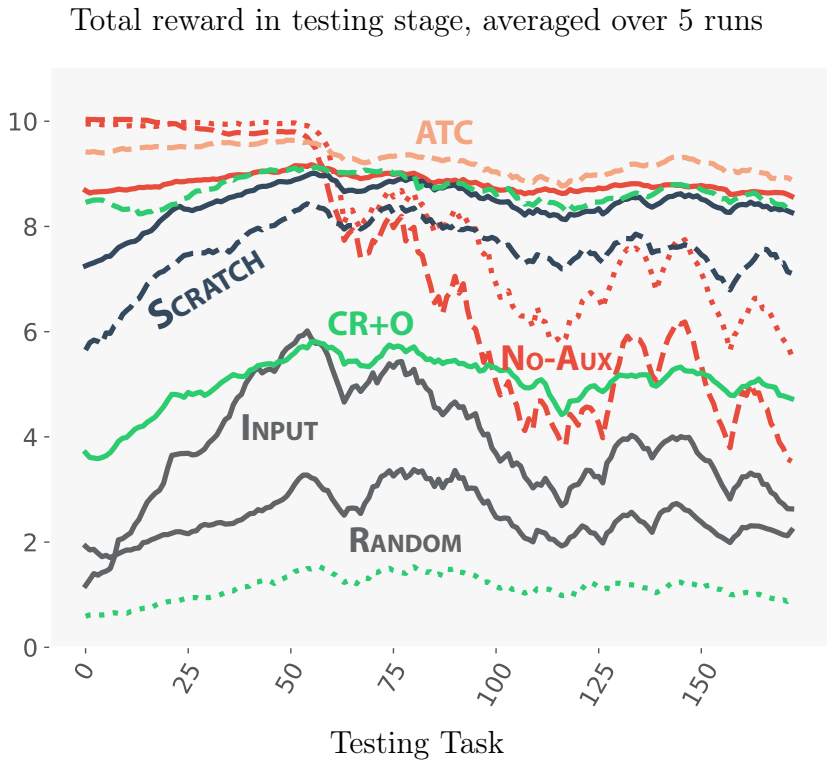

Figure 6.3: Comparing generalization performance of CR+O representations with the relevant baselines. The tasks on the x-axis are arranged by similarity to the training task: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. Dashed lines, dotted lines, and solid lines represent ReLU, ReLU(L), and FTA representations respectively.

to their extreme, and it greatly reduces dynamics awareness. These changes in properties can clearly explain the reduction in generalization performance, especially in the case of ReLU(L) where orthogonality and sparsity are almost 1. In the case of ReLU, on the other hand, this loss helped to increase the value of orthogonality from 0 and all other properties are slightly increased. These results confirm our previous claim that representations with high dynamics awareness, diversity, complexity reduction, and interim level of orthogonality generalize well. Interestingly, diversity is also increased in all cases, further confirming the strong linear correlation between this property and complexity reduction stated in Section B.2.4 in Appendix B.

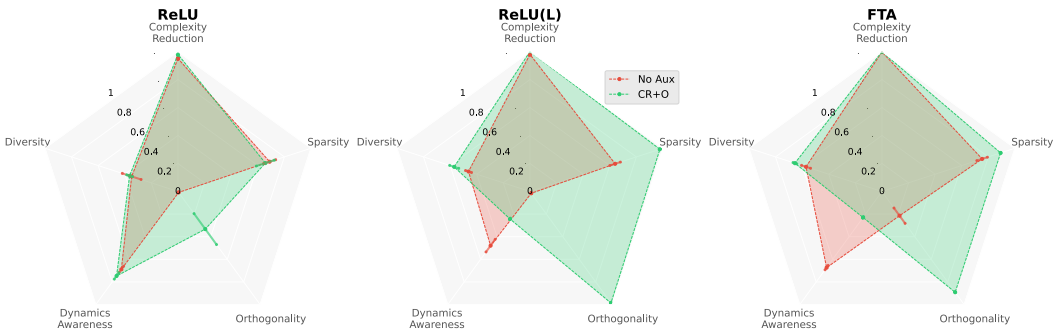

Figure 6.4: Properties of CR+O representations. The CR+O task produces ReLU(L) and FTA representations that do not generalize well and ReLU representations that do. Each subplot shows five property values achieved by six different representations: ReLU, ReLU(L), or FTA with and without CR+O. This auxiliary task significantly changes dynamics awareness and orthogonality. We did not include non-interference as CR+O had no impact on it.

6.3 Discussion

The findings in this chapter have two key implications. First, our results provide additional evidences for the conclusions made in Chapter 5. These evidences are stronger than those provided before because the question is asked in a more directed way. We specifically addressed this question: whether explicit maximization of the desirable properties in representations improve their generalization performance. We clearly saw that maximization of the desirable properties results in better generalization performance, unless doing so harms the value of other properties that are critical for achieving good generalization performance. Second, we showed that the use of property-based auxiliary losses, notably DA+O loss, significantly improves generalization performance. This suggests that property-based auxiliary losses, including the losses introduced in this chapter, are a promising future direction to improve the generalization performance of the current deep reinforcement learning algorithms.

Chapter 7

Conclusion

The goal of this work is to make progress towards an answer to a classic question: how do the properties of representations, which emerge under standard architectures used in deep reinforcement learning, relate to the generalization performance? We introduced a method to measure the similarity between training and testing tasks and designed experiments to evaluate learned representations (Chapter 4). All tasks are similar in that they involve navigating to locations in the same Maze. Intuitively, successful generalization should be possible even to locations that are quite far from the goal in the training task. We found that

1. Most ReLU-based representations generalized well only to very similar tasks, potentially highlighting why successful generalization (specifically representation transfer) has been so difficult in reinforcement learning (the vast majority of SOTA agents use ReLU networks).

2. ATC and VirtualVF5 were the only ReLU-based representations that generalized well across similar and dissimilar tasks.

3. ATC's generalization performance suffers greatly if data augmentation is eliminated, showing that it is an essential part of this contrastive loss.

4. Simple data augmentation techniques on top of No-Aux network can greatly improve generalization performance, indicating it might be a promising direction for future work.

5. FTA activation significantly improved generalization performance, suggesting it might be a promising activation to use going forward.

6. Successful generalization was not possible with linear value functions, even in this seemingly simple environment (Chapter 5).

We extensively and systematically investigated the properties of all these (good and bad) representations attempting to better understand what causes the improvement in generalization performance (Chapter 5). We defined diversity, complexity reduction, and dynamics awareness, as well as used measures of orthogonality, sparsity, and non-interference from the literature (Chapter 3). In general, interim values for properties were better: representations at the very extremes were never the best. Further, we found that the best representations maintained high capacity (complexity reduction, dynamics awareness, and diversity), and lower orthogonality. These conclusions do not mean representations should have low orthogonality, for example, but rather representations that emerge under training with auxiliary losses tend to do more poorly if orthogonality is higher or if it is very small (at the extreme) in our particular generalization setting. These findings allowed us to compare the properties of individual representations and use them to explain the difference between their generalization performance.

Taking inspiration from our findings, we designed two auxiliary losses to encourage specific properties in the representation (Chapter 6). Specifically, these property-based auxiliary losses maximize two pairs of properties: dynamics awareness and orthogonality, and complexity reduction and orthogonality. We demonstrated how such losses significantly improve small-sized ReLU representations (i.e., representations with a low number of features), achieving competitive generalization performance to our best baselines. Analyzing their properties, we confirm that representations with high dynamics awareness, diversity, complexity reduction, and an interim level of orthogonality successfully generalize.

This thesis investigated only one relatively simple environment (a Maze) and one neural network architecture, albeit it is an environment with many

different tasks/goals. This network architecture is one of the most widely used in deep reinforcement learning. Even in just this setting, there was a mountain of data to analyze. To gain insight into the complex representations learned by our agents, it was necessary to start in a simple setting and develop a clear and systematic methodology. A natural next step is to execute the same procedure in other, possibly larger, environments; or to use different neural network architectures. We would like to highlight, however, that the results even in this one setting are already informative and change our perspective on these representations. A priori, one might have thought that successful generalization would be very easy in this environment; after all, we are not learning small networks here! Yet, we have repeatedly hit roadblocks.

The results here highlight how difficult it can be to achieve successful generalization, specifically in a setting where we have a single task to train on. The variety in agent performance obtained here, and the significant changes in performance when moving from closer to further goal locations, already allows us to tease apart differences in approaches and properties. The specific conclusions about network architectures and activations, auxiliary losses, and even properties, may be different in other environments, but the higher-level conclusions about the relevance of these properties, the interactions between components, and the need for a careful methodology to understand these nuances will extend.

In this work, we saw that maximizing different sets of properties, using property-based auxiliary losses, significantly improves generalization performance. We explored only two different combinations of these properties; any other combinations are left to be explored. We speculate that the main reason for the failure of FTA and ReLU(L)-based CR+O representations was low dynamics awareness. So, one promising direction would be to add dynamics awareness as part of this loss. There are also many different ways to design these auxiliary losses. For instance, we can maximize dynamics awareness using a contrastive framework [e.g., 71] instead of maximizing it directly in the representation. Lastly, the efficacy of these losses is limited to this generalization setting, so one natural next step would be to test them in more complex

generalization settings such as ProcGen [9].

There are a variety of generalization settings for which we can use the methodology proposed in this work. There has been substantial effort to characterize generalization and overfitting in deep reinforcement learning, primarily in terms of performance [10, 15, 53, 60]. Notably, prior work illustrated it is possible to successfully generalize across Atari modes [15], but did not yet quantify any properties of those representations. A natural next step is to revisit these experiments with new tools to understand and improve the representations in these benchmark environments. The computational effort required to do this in a systematic and principled way will be a major challenge.

References

[1] R. Agarwal, M. C. Machado, P. S. Castro, and M. G. Bellemare, "Contrastive Behavioral Similarity Embeddings for Generalization in Reinforcement Learning," in *International Conference on Learning Representations*, 2021. 11, 17

[2] K. Asadi, D. Misra, and M. Littman, "Lipschitz Continuity in Model-Based Reinforcement Learning," in *International Conference on Machine Learning*, 2018. 21

[3] A. Atrey, K. Clary, and D. Jensen, "Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2020. 3

[4] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver, "Successor Features for Transfer in Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2017. 14, 71

[5] M. Bellemare, W. Dabney, R. Dadashi, A. A. Taiga, P. S. Castro, N. Le Roux, D. Schuurmans, T. Lattimore, and C. Lyle, "A Geometric Perspective on Optimal Representations for Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2019. 3, 12, 13

[6] E. Bengio, J. Pineau, and D. Precup, "Interference and Generalization in Temporal Difference Learning," in *International Conference on Machine Learning*, 2020. 24

[7] R. Caruana, "Multitask Learning," *Machine learning*, 1997. 3

[8] W. Chung, S. Nath, A. Joseph, and M. White, "Two-Timescale Networks for Nonlinear Value Function Approximation," in *International Conference on Learning Representations*, 2019. 13, 36

[9] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging Procedural Generation to Benchmark Reinforcement Learning," in *International Conference on Machine Learning*, 2020. 17, 58

[10] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying Generalization in Reinforcement Learning," in *International Conference on Machine Learning*, 2019. 58

[11] T. Dai, K. Arulkumaran, S. Tukra, F. Behbahani, and A. A. Bharath, "Analysing Deep Reinforcement Learning Agents Trained with Domain Randomisation," *arXiv preprint arXiv:1912.08324*, 2019. 3

[12] P. Dayan, "Improving Generalization for Temporal Difference Learning: The Successor Representation," *Neural Computation*, 1993. 14

[13] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang, "Is a Good Representation Sufficient for Sample Efficient Reinforcement Learning?" In *International Conference on Learning Representations*, 2019. 1

[14] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems*, 1990. 2

[15] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and Regularization in DQN," *arXiv preprint arXiv:1810.00123*, 2018. 17, 38, 58

[16] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *International Conference on Machine Learning*, 2017. 3

[17] C. Finn, X. Y. T. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep Spatial Autoencoders for Visuomotor Learning," in *IEEE International Conference on Robotics and Automation*, 2016. 3

[18] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau, "Combined Reinforcement Learning via Abstract Representations," in *AAAI Conference on Artificial Intelligence*, 2019. 3, 13

[19] R. M. French, "Catastrophic Forgetting in Connectionist Networks," *Trends in Cognitive Sciences*, 1999. 3

[20] S. Gamrian and Y. Goldberg, "Transfer Learning for Related Reinforcement Learning Tasks via Image-to-Image Translation," in *International Conference on Machine Learning*, 2019. 17

[21] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare, "DeepMDP: Learning Continuous Latent Space Models for Representation Learning," in *International Conference on Machine Learning*, 2019. 36

[22] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *International Conference on Artificial Intelligence and Statistics*, 2010. 67

[23] L. Graesser, U. Evci, E. Elsen, and P. S. Castro, "The State of Sparse Training in Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2022. 19

[24] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and Understanding Atari Agents," in *International Conference on Machine Learning*, 2018. 3

[25] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive Mapping and Planning for Visual Navigation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3

[26] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "DARLA: Improving Zero-Shot Transfer in Reinforcement Learning," in *International Conference on Machine Learning*, 2017. 3, 36

[27] G. Z. Holland, E. J. Talvitie, and M. Bowling, "The Effect of Planning Shape on Dyna-Style Planning in High-Dimensional State Spaces," *arXiv preprint arXiv:1806.01825*, 2018. 1

[28] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement Learning with Unsupervised Auxiliary Tasks," in *International Conference on Learning Representations*, 2017. 3, 12–14, 36, 38

[29] K. Javed and M. White, "Meta-Learning Representations for Continual Learning," in *Advances in Neural Information Processing Systems*, 2019. 3

[30] P. Kanerva, *Sparse Distributed Memory*. MIT press, 1988. 2

[31] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations*, 2015. 30, 68, 71

[32] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning," *arXiv preprint arXiv:2111.09794*, 2021. 1, 16

[33] V. R. Kompella, M. Luciw, and J. Schmidhuber, "Incremental Slow Feature Analysis: Adaptive Low-Complexity Slow Feature Updating from High-Dimensional Input Streams," *Neural Computation*, 2012. 2

[34] G. Konidaris, S. Osentoski, and P. Thomas, "Value Function Approximation in Reinforcement Learning Using the Fourier Basis," in *AAAI Conference on Artificial Intelligence*, 2011. 2

[35] T. Kurutach, A. Tamar, G. Yang, S. J. Russell, and P. Abbeel, "Learning Plannable Representations with Causal Infogan," in *Advances in Neural Information Processing Systems*, 2018. 3

[36] C. L. Lan, S. Tu, A. Oberman, R. Agarwal, and M. G. Bellemare, "On the generalization of representations in reinforcement learning," in *International Conference on Artificial Intelligence and Statistics*, 2022. 18

[37] S. Lange and M. Riedmiller, "Deep Auto-encoder Neural Networks in Reinforcement Learning," in *International Joint Conference on Neural Networks*, 2010. 11

[38] M. Laskin, A. Srinivas, and P. Abbeel, "Curl: Contrastive Unsupervised Representations for Reinforcement Learning," in *International Conference on Machine Learning*, 2020. 11

[39] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement Learning with Augmented Data," in *Advances in Neural Information Processing Systems*, 2020. 11

[40] T. Lattimore, C. Szepesvari, and G. Weisz, "Learning With Good Feature Representations in Bandits and in RL with a Generative Model," in *International Conference on Machine Learning*, 2020. 1

[41] L. Le, R. Kumaraswamy, and M. White, "Learning Sparse Representations in Reinforcement Learning with Sparse Coding," in *International Joint Conference on Artificial Intelligence*, 2017. 19

[42] E. Lecarpentier, D. Abel, K. Asadi, Y. Jinnai, E. Rachelson, and M. L. Littman, "Lipschitz Lifelong Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2021. 21

[43] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," *arXiv preprint arXiv:2005.01643*, 2020. 34

[44] V. Liu, R. Kumaraswamy, L. Le, and M. White, "The Utility of Sparse Representations for Control in Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2019. 3, 19, 24

[45] V. Liu, A. White, H. Yao, and M. White, "Towards a Practical Measure of Interference for Reinforcement Learning," *arXiv preprint arXiv:2007.03807*, 2020. 24

[46] M. C. Machado, A. Barreto, and D. Precup, "Temporal Abstraction in Reinforcement Learning with the Successor Representation," *arXiv preprint arXiv:2110.05740*, 2021. 31

[47] M. C. Machado, M. G. Bellemare, and M. Bowling, "Count-Based Exploration with the Successor Representation," in *AAAI Conference on Artificial Intelligence*, 2020. 14

[48] M. C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, and M. Campbell, "Eigenoption Discovery through the Deep Successor Representation," in *International Conference on Learning Representations*, 2018. 13, 14, 21

[49] S. Mahadevan and M. Maggioni, "Proto-Value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes," *Journal of Machine Learning Research*, 2007. 2

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-Level Control through Deep Reinforcement Learning," *Nature*, 2015. 3, 4, 10, 68

[51] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-Conditional Video Prediction Using Deep Networks in Atari Games," in *Advances in Neural Information Processing Systems*, 2015. 12

[52] J. Oh, S. Singh, and H. Lee, "Value Prediction Network," in *Advances in Neural Information Processing Systems*, 2017. 3, 13

[53] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing Generalization in Deep Reinforcement Learning," *arXiv preprint arXiv:1810.12282*, 2018. 58

[54] C. Painter-Wakefield and R. Parr, "Greedy Algorithms for Sparse Reinforcement Learning," in *International Conference on Machine Learning*, 2012. 2

[55] Y. Pan, K. Banman, and M. White, "Fuzzy Tiling Activations: A Simple Approach to Learning Sparse Representations Online," in *International Conference on Learning Representations*, 2021. 3, 11, 12, 19, 24

[56] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, "An Analysis of Linear Models, Linear Value-Function Approximation, and Feature Selection for Reinforcement Learning," in *International Conference on Machine Learning*, 2008. 13

[57] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing Feature Generation for Value-Function Approximation," in *International Conference on Machine Learning*, 2007. 2

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019. 68

[59] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-Driven Exploration by Self-Supervised Prediction," in *International Conference on Machine Learning*, 2017. 13

[60] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards Generalization and Simplicity in Continuous Control," in *Advances in Neural Information Processing Systems*, 2017. 58

[61] B. Ratitch and D. Precup, "Sparse Distributed Memories for On-Line Value-Based Reinforcement Learning," in *European Conference on Machine Learning*, 2004. 3

[62] R. J. Rummel, *Applied factor analysis*. Northwestern University Press, 1988. 23

[63] C. Rupprecht, C. Ibrahim, and C. J. Pal, "Finding and Visualizing Weaknesses of Deep Reinforcement Learning Agents," in *International Conference on Learning Representations*, 2020. 3

[64] D. Russo and B. Van Roy, "Eluder Dimension and the Sample Complexity of Optimistic Exploration," in *Advances in Neural Information Processing Systems*, 2013. 1

[65] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *Nature*, 2020. 3, 13

[66] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. C. Courville, and P. Bachman, "Data-Efficient Reinforcement Learning with Self-Predictive Representations," in *ICLR*, 2021. 21

[67] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris, "The Predictron: End-to-End Learning and Planning," in *International Conference on Machine Learning*, 2017. 3

[68] S. Sinha, H. Bharadhwaj, A. Srinivas, and A. Garg, "D2rl: Deep dense architectures in reinforcement learning," *arXiv preprint arXiv:2010.09163*, 2020. 11

[69] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal Planning Networks: Learning Generalizable Representations for Visuomotor Control," in *International Conference on Machine Learning*, 2018. 3, 13

[70] K. L. Stachenfeld, M. Botvinick, and S. J. Gershman, "Design Principles of the Hippocampal Cognitive Map," in *Advances in Neural Information Processing Systems*, 2014. 21

[71] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling Representation Learning from Reinforcement Learning," in *International Conference on Machine Learning*, 2021. 11, 12, 15, 17, 21, 57, 70

[72] F. P. Such, V. Madhavan, R. Liu, R. Wang, P. S. Castro, Y. Li, J. Zhi, L. Schubert, M. G. Bellemare, J. Clune, and J. Lehman, "An Atari Model Zoo for Analyzing, Visualizing, and Comparing Deep Reinforcement Learning Agents," in *International Joint Conference on Artificial Intelligence*, 2019. 3

[73] R. S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," in *Advances in Neural Information Processing Systems*, 1996. 2, 3, 19

[74] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018. 1, 2

[75] C. Szepesvári, *Algorithms for Reinforcement Learning*. Morgan and Claypool, 2010. 13

[76] E. Talvitie, "Self-Correcting Models for Model-Based Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2017. 1

[77] S. Thrun, "Is Learning the n-th Thing any Easier than Learning the First?" In *Advances in Neural Information Processing Systems*, 1996. 3

[78] J. Tyo and Z. Lipton, "How Transferable Are the Representations Learned by Deep Q Agents?" *arXiv preprint arXiv:2002.10021*, 2020. 38

[79] M. Vischer, R. T. Lange, and H. Sprekeler, "On Lottery Tickets and Minimal Task Representations in Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2022. 19

[80] Y. Wan, M. Zaheer, A. White, M. White, and R. S. Sutton, "Planning with Expectation Models," in *International Joint Conference on Artificial Intelligence*, 2019. 1

[81] H. Wang, "Emergent Representations in Reinforcement Learning and Their Properties," Ph.D. dissertation, University of Alberta, 2020. iii, 23

[82] H. Wang, E. Miahi, M. White, M. C. Machado, Z. Abbas, R. Kumaraswamy, V. Liu, and A. White, "Investigating the Properties of Neural Network Representations in Reinforcement Learning," *arXiv preprint arXiv:2203.15955*, 2022. iii

[83] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2016. 11

[84] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images," in *Advances in Neural Information Processing Systems*, 2015. 3

[85] M. White, "Unifying Task Specification in Reinforcement Learning," in *International Conference on Machine Learning*, 2017. 9

[86] W. Whitney, R. Agarwal, K. Cho, and A. Gupta, "Dynamics-Aware Embeddings," in *International Conference on Learning Representations*, 2020. 19

[87] S. Witty, J. K. Lee, E. Tosch, A. Atrey, M. Littman, and D. Jensen, "Measuring and Characterizing Generalization in Deep Reinforcement Learning," *arXiv preprint arXiv:1812.02868*, 2018. 17

[88] Y. Wu, G. Tucker, and O. Nachum, "The Laplacian in RL: Learning Representations with Efficient Approximations," in *International Conference on Learning Representations*, 2019. 3, 15

[89] G. Yang, A. Zhang, A. Morcos, J. Pineau, P. Abbeel, and R. Calandra, "Plan2Vec: Unsupervised Representation Learning by Latent Plans," in *Conference on Learning for Dynamics and Control*, 2020. 13

[90] M. Yang and O. Nachum, "Representation Matters: Offline Pretraining for Sequential Decision Making," in *International Conference on Machine Learning*, 2021. 36

[91] D. Yarats, I. Kostrikov, and R. Fergus, "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels," in *International Conference on Learning Representations*, 2021. 11, 12

[92] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the Black Box: Understanding DQNs," in *International Conference on Machine Learning*, 2016. 3

[93] A. Zhang, N. Ballas, and J. Pineau, "A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning," *arXiv preprint arXiv:1806.07937*, 2018. 17

[94] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A Study on Overfitting in Deep Reinforcement Learning," *arXiv preprint arXiv:1804.06893*, 2018. 17

[95] H. Zhao, J. Wu, Z. Li, W. Chen, and Z. Zheng, "Double Sparse Deep Reinforcement Learning via Multilayer Sparse Coding and Nonconvex Regularized Pruning," *IEEE Transactions on Cybernetics*, 2022. 19

[96] Z. Zhu, K. Lin, and J. Zhou, "Transfer Learning in Deep Reinforcement Learning: A Survey," *arXiv preprint arXiv:2009.07888*, 2022. 16

Appendix A

Experimental Hyperparameters

In this chapter, we first discuss the hyperparameters that are shared across all our experiments. Then, we describe how each of the properties is measured in detail. At last, we go over the implementation and various hyperparameters used to train each of the auxiliary networks.

A.1 General Hyperparameter Setting

We use the same neural network architecture for the representation and value network in all of our experiments, except for the representation layer (i.e., the final layer of the representation network). For the representation function, we use a 2-layer convolutional network followed by the representation layer. The first convolutional layer has kernel size of 4, stride of 1, padding of 1, and 32 channels, and the second one has kernel size of 4, stride of 2, padding of 2, and 16 channels on the second layer. The representation layer is a fully-connected layer that differs across experiments based on the type of activation function. Table A.1 shows how the number of nodes and features differs across activations. All hidden layers are initialized using Xavier initialization [22]. For the target network, we use two fully-connected layers, each having 64 neurons and using ReLU.

Regarding the FTA setting, we use bins 20 with upper and lower bounds equal to 2 and -2. We tested η of $0.2, 0.4, 0.6$, and 0.8 for the no auxiliary task agent, and we fixed $\eta = 0.2$ for agents trained with auxiliary task.

During the training stage, the input images (states) are normalized to be

Table A.1: Number of features and nodes of the representation for each activation.

Activation	Last Hidden Layer Nodes	Features
ReLU	32	32
ReLU(L)	640	640
FTA	32	640

in the range $[-1, 1]$, using $\frac{2\boldsymbol{S}}{255} - 1$, where $\boldsymbol{S}$ is the 15×15 input image. We use the Adam optimizer to update the weights [31], and we used the mean-squared error as the value network's loss. The batch size is set to 32. The buffer has length $10,000$. A target network was used with the synchronization frequency set to 64. The buffer's memory size was $100,000$ and there were 32 samples randomly chosen at each step. The agent learns for $300,000$ steps with ϵ-greedy. We use the same hyperparameters for the testing stage, except that all agents, including baselines, learned for $100,000$ steps only.

The learning rate was swept for every representation learning architecture and control task. The best setting was picked according to the averaged performance over 5 runs. Each run uses a different random seed. In the training stage and for the baseline agents that learn from scratch in the testing stage, we swept the learning rate over $[0.001, 0.0003, 0.0001, 0.00003, 0.00001]$. In the testing phase, we swept over $[0.01, 0.003, 0.001, 0.0003, 0.0001]$ for representations with 32 features and over $[0.001, 0.0003, 0.0001, 0.00003, 0.00001]$ for the representations with 640 features.

We use a fixed $\epsilon = 0.1$ during all stages, except during the testing phase of the linear value functions. In this case, it turned out to be harder for the agent to converge when keeping other hyperparameters the same as in the non-linear value function. Therefore, we use a scheduling technique (the same as the one used in the DQN paper [50]) that reduces the value of ϵ over the first $100,000$ steps from 1 to 0.1.

All our experiments are conducted using Python and the PyTorch framework [58].

A.2 Measuring Representation Properties

Overall, we have a dataset of $1,000$ transitions generated by a random walk and use them to measure six representation properties, described in Chapter 3. In practice, each of these properties is measured in a different way throughout training to make sure that the measurement is computationally cheap and at the same time accurate. These properties are measured every $10,000$ steps during training. We explain how each of these properties are measured below.

Complexity Reduction and **Diversity** are measured in the same way. To compute these properties, we randomly sample 100 transitions from the dataset and compute this measure on these transitions. We repeat this procedure for 10 times and report the average over these values as a measure of these properties. We take this approach because computing this measure over $1,000$ transitions is computationally expensive.

Measuring **Dynamics Awareness** requires two different ways of sampling: one for measuring the distance between adjacent states and one for measuring the distance between non-adjacent states. For the former, we consider the whole dataset, measuring the distance between the representation of each state and its next state. For the latter, we randomly pair $1,000$ states without replacement and compute the distance between each pair of representations.

To compute **Orthogonality**, we first draw a sample of 100 transitions from the dataset and compute the orthogonality measure between all unique pairs of these transitions, excluding the pairs with the same transition ($9,900$ pairs). We repeat this process 10 times and report the averaged value. **Sparsity**, on the other hand, is computed in a simple manner. We generate the representations of each state in the dataset and compute the sparsity of all these representations.

To compute **non-interference**, we need to compute **Update Interference** every $k = 64$ timesteps, i.e., target network update frequency. We randomly draw 32 samples (the same as batch size) from the dataset every $k = 64$ timesteps (i.e., target update frequency) and use them to compute Update Interference. Then, the non-interference is measured every $10,000$ iterations by

considering 90^{th} percentile of Update Interference calculated in the previous steps.

A.3 Implementation Details of Auxiliary Tasks

In this section, we provide a more detailed explanation of each of the auxiliary tasks used in this thesis. The loss functions for these tasks are described in Chapter 2. All losses use the same samples from the replay buffer to update the value function, except for ATC, LAPLACIAN, DA+O, and CR+O. We use a neural network with two fully-connected layers (each having 64 nodes) as an auxiliary function, $f_{\boldsymbol{\theta_A}}$, for all losses, except ATC, IR, LAPLACIAN, DA+O, and CR+O.

The **ATC** loss encourages the network to learn similar representations for an input state, s_t, with one from a pre-determined, near-future time step input state, s_{t+k}. We consider $k = 3$. In contrast to other losses used in this work, this loss uses more than one auxiliary head to compute itself. To do so, it first applies a data augmentation technique called random shift with a probability of 0.1 and padding of 4 to both of these input states. Then, it feeds the augmented version of s_t, $\text{AUG}(s_t)$, through a set of networks to compute p_t, where $f_{\boldsymbol{\theta_A}}$ is a linear mapping of representation into an *embedding space* with a size of 32, and $f_{\boldsymbol{\theta_C}}$ is a single layer neural network with a hidden layer size of 64, and an output size of 32. Then, $\text{AUG}(s_{t+k})$ is fed into a momentum encoder of $\phi_{\boldsymbol{\theta_R}}$ ($\phi_{\breve{\boldsymbol{\theta}}_{\boldsymbol{R}}}$) and $f_{\boldsymbol{\theta_A}}$ ($f_{\breve{\boldsymbol{\theta}}_{\boldsymbol{A}}}$) to compute c_{t+k}. The output of these networks, p_t and c_{t+k}, are combined with each other through a 32×32 matrix called $\boldsymbol{W}$ to compute the ATC loss. In contrast to the original paper that uses a complex learning-rate scheduling technique, we implement this loss in its simplest form by using a fixed learning-rate. This learning-rate is swept across the values $[0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001]$. We update the momentum encoder in every step using a moving average step-size of $\tau = 0.01$ [cf. 71]. Interestingly, we swept over random shift values of $[0, 0.1, 0.01, 0.2, 0.3]$, the embedding space size of $[8, 16, 32, 64, 128]$, and k of $[2, 3, 4, 5]$, and found that the hyperparameters used in the ATC paper worked

Table A.2: Selected values of property-based auxiliary losses.

Activation	$\beta_{\text{DA+O}}$	$w_{\text{DA+O}}$	$\beta_{\text{CR+O}}$	$w_{\text{CR+O}}$
ReLU	5	1	0.1	0.001
ReLU(L)	20	0.1	5	1
FTA	32	0.0001	5	1

best. The batch size is the same as the batch size used to calculate the loss of the value function, and the weight of this auxiliary loss is set to 1.

The **Laplacian** is applied directly to the output of the representation layer. In other words, its auxiliary head is the identity function. We take separate batches of size 32 from the replay buffer to compute each of the two expectations used to compute this loss. We use different w_{aux} and β for each of the activation functions. For ReLU, we use $\beta = 1$ and $w_{aux} = 0.1$. For ReLU(L), we use $\beta = 5$ and $w_{aux} = 0.1$. For FTA, we use $\beta = 1.0$ and $w_{aux} = 0.00001$. We report the main results using the same $\gamma_L = 0.9$ value as in the original paper. We also look at γ_L of $[0, 0.1, 0.25, 0.75, 0.99]$ to see how much this auxiliary loss can help the ReLU-based representations when these values are low and high.

IR task reconstructs the input image from the representation through a deconvolutional network [cf. 31]. In the auxiliary head, the representation is first projected into a hidden layer with 1024 nodes then sent to a 2-layer deconvolutional neural network, with the first layer having 4 kernels, 32 channels, strides of 2, padding of 2 and the second one having kernel size of 32, 3 channels, strides of 1, pad size of 1. The output of the last layer has the same size as the input image (that is, $15 \times 15 \times 3$). The weight of this auxiliary loss was set to 0.0001.

SF uses an extra head (a fully-connected layer with a single node) to predict the reward linearly from the representation ϕ_t. This helps to satisfy the property of the successor features: being able to predict the reward using a linear mapping [cf. 4]. We set γ_{SF} to 0.99 and use the weight of 1. We use the same weight for the **Reward** task, and the weight of 0.001 for **NAS**.

To compute the loss for **XY**, we store the agent's coordinates in the envi-

ronment in the replay buffer during the learning. We use a weight of 0.0001. For **VirtualVF-1**, we use a single head to learn a value function for the goal state on the grid $[7, 7]$. For **VirtualVF-5**, we use five auxiliary heads to learn value functions for five goal states separately. The locations of the goal states are $[0, 0]$, $[0, 14]$, $[14, 0]$, $[14, 14]$, and $[7, 7]$. The weight of both of these auxiliary tasks remained 1 but the discount rate on the auxiliary head is set to be lower than the training task ($\gamma_{\text{VF}} = 0.9$). In this way, the agent can focus on the main task. All auxiliary heads use the same network structure as XY.

CR+O and **DA+O** do not require an auxiliary head and are applied directly to the representation layer, like the Laplacian loss. These losses use a special weight to adjust the contribution of orthogonality to the representation updates called β. During our experiments, we sweep over the values $[20, 10, 5, 1, 0.1, 0.01]$ of β and choose the highest value that results in successful convergence in the training task.

Appendix B

Additional Experimental Results & Analysis

In this chapter, we present the additional results generated by our experiments. We first present the generalization performance of the representations discussed in Chapter 5 in more detail. Specifically, we show how ATC performs without data augmentation technique. Then, we present representation properties of individual representations, discussing why they perform as they do. We also talk about the complexity reduction of all representations in more detail.

B.1 Generalization Performance

During the training state, we make sure that the training task is fully solved, i.e., agent is able to reach the goal state in 100 timesteps. After the agent converges to the best solution in the training task, we save the representation. We show the learning curves of all representation learning architectures in Figure B.1. This figure shows that the early saved representations have converged to the highest possible return of 1. We use these saved representations to evaluate the generalization performance in the testing phase. In the remainder of this section, we describe generalization performance of different representations that emerged through this procedure.

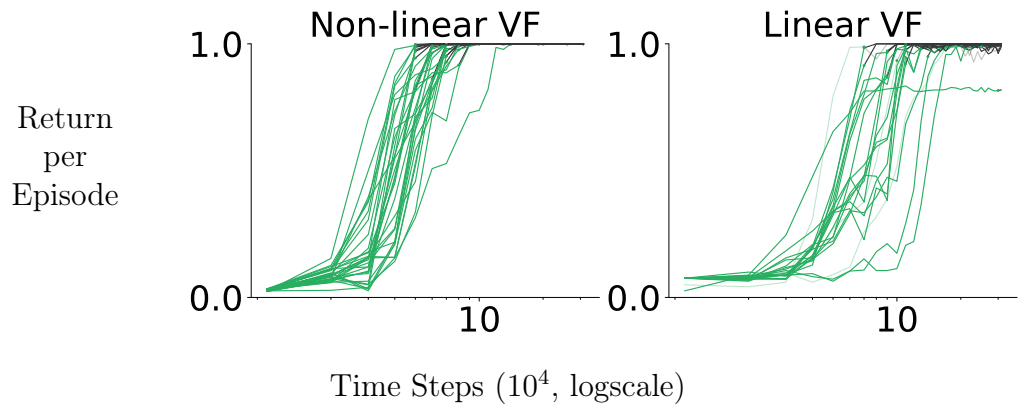

Figure B.1: Learning-curves of representations learned in the training stage. Return converges in the representation training stage. The plot shows the averaged return over the most recent 100 episodes at each checkpoint. The x-axis is the number of time steps and the y-axis is the average return. Each curve represents one agent specification (activation and auxiliary task pair). As our main focus is not to compare the learning efficiency during the representation learning step, and the difference between learning curves is not large, we only show the general trend by plotting every curve with the same color. The curve changes color to black, at the time point where we took the representation and fixed it.

B.1.1 Generalization Performance of Larger ReLU

We show the generalization performance of ReLU(L) in Figure B.2. The ReLU(L) setting stays between ReLU and FTA representations: ReLU(L) keeps the same activation function as ReLU representation, but increases the size of the representation layer to 640, which is the same as the size of FTA representations. Therefore, it maintains the same value function capacity as the FTA representations. The pattern in the generalization performance of ReLU(L) is similar to ReLU. As the testing tasks become dissimilar, the generalization performance drops below the Scratch agent. In general, when considering the total reward obtained by the agent, the performance of ReLU(L) is better than ReLU and worse than FTA.

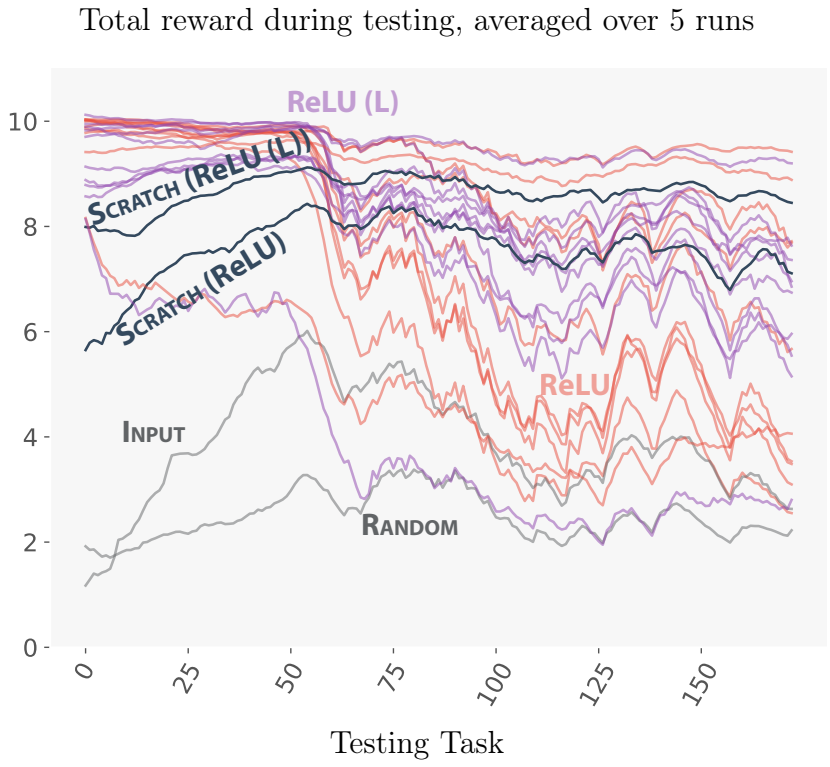

Figure B.2: Generalization performance of larger ReLU representations (50 in total) compared to ReLU(L) representations (50 in total) on 173 testing tasks. The tasks on the x-axis are arranged by similarity to training tasks: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. The black line shows the performance when learning in each transfer task from scratch, with the same representation size as ReLU(L). Lines completely above the black line indicate a representation yielded successful transfer in all tasks. Lines that start above the black line but fall below it as we move left to right indicate a representation that transfers to similar tasks but not dissimilar tasks.

B.1.2 Generalization Performance of Individual ReLU Representations

Generalization performance of ReLU-based representations are presented in Figure B.3. The only representations that consistently obtained good performance in all testing tasks were ReLU+ATC and ReLU+VirtualVF5. Although ReLU+XY and ReLU+VirtualVF1 obtained strong generalization performance on dissimilar testing tasks, their generalization performance on testing tasks was not as good. Interestingly, the Laplacian representation achieves the worst performance among all the representations studied in this

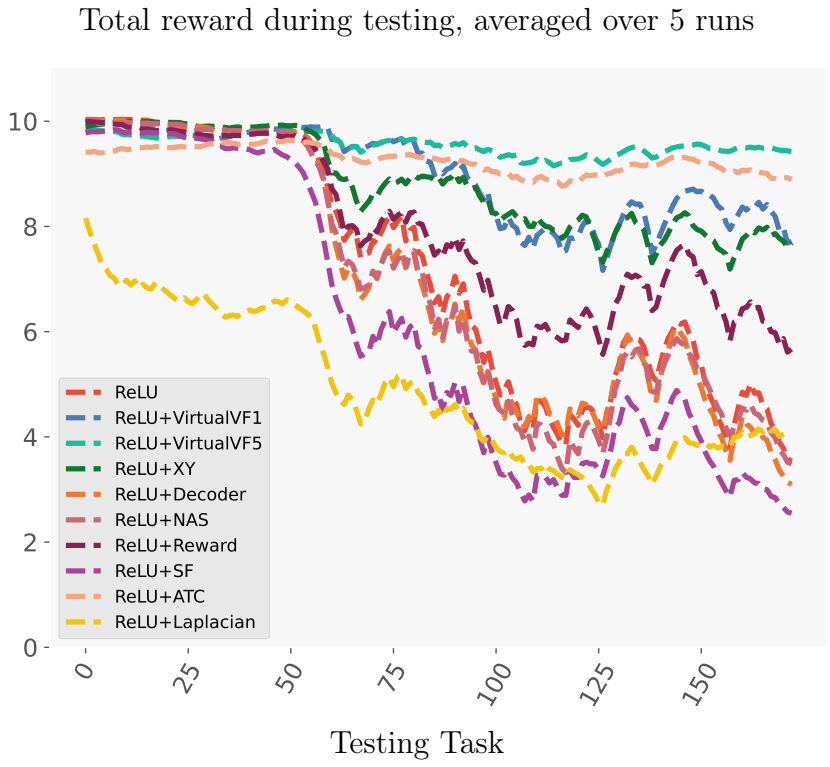

Figure B.3: Generalization performance of ReLU representations (50 in total) in 173 testing tasks. The tasks on the x-axis are arranged by similarity to training tasks: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. The black line shows the performance when learning in each testing task from scratch, with the same representation size as ReLU(L). Lines completely above the black line indicate a representation successfully generalizes in all tasks. Lines that start above the black line but fall below it as we move left to right indicate a representation that generalizes well to similar tasks but not to dissimilar tasks.

thesis, failing to generalize well both in similar and dissimilar tasks.

B.1.3 Generalization Performance of Different Auxiliary Tasks and Activation Fuctions

Figure B.4 shows the 95% confidence interval of generalization performance with different representation sizes, activation functions, and different auxiliary tasks, with a non-linear value function. This plot shows trends similar to Figure 5.3 presented in Chapter 5.

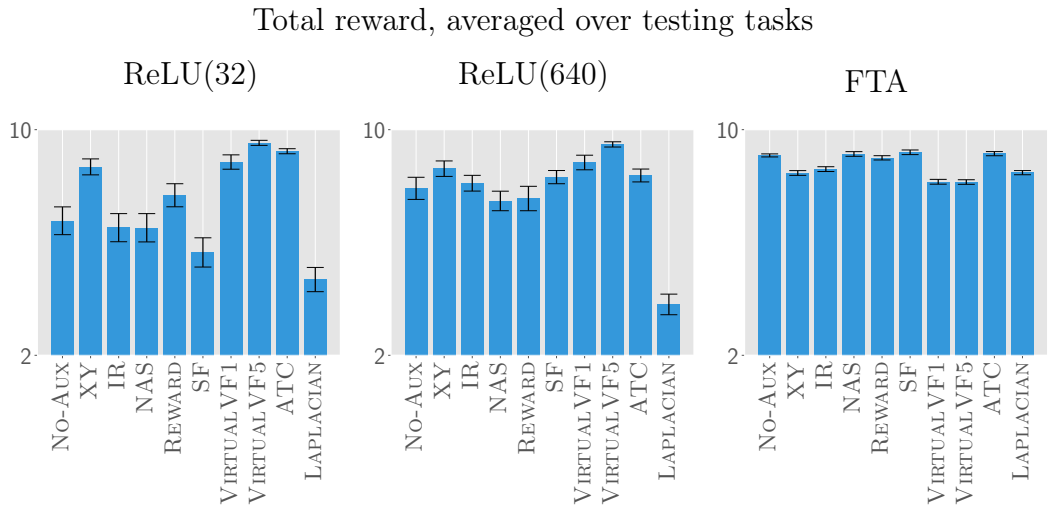

Figure B.4: Generalization performance of different auxiliary tasks and activation functions. Generalization performance depends on the activation function, representation size, and auxiliary tasks. This plot presents the same data as Figure 5.3, but the error bar shows a 95% confidence interval. The bar shows the mean value over 5 seeds $\times$ 173 testing tasks.

B.1.4 The Importance of Data Augmentation for ATC

The results for ATC with and without the usage of data augmentation technique is shown in Figure B.5. The generalization performance of ReLU degrades greatly as we remove data augmentation from ATC. This shows how crucial data augmentation usage is for ATC. These results further show that using the data augmentation technique on top of the ReLU-based representations without any auxiliary tasks results in performance similar to ATC. These findings raise the question of whether the contrastive loss architecture is responsible for the performance of ATC or only the data augmentation technique.

B.2 Representation Properties

In this section, we first study the representation properties of specific representations, namely ATC and Laplacian. Then, we take a closer look at the complexity reduction of different representations. Lastly, we discuss the connection between the six properties that are discussed in this work.

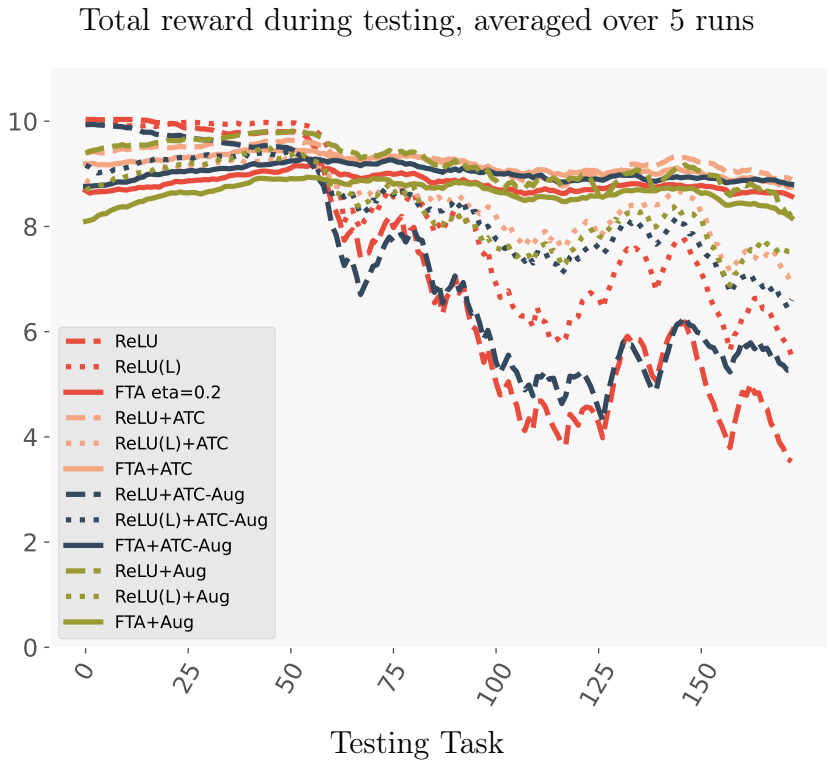

Figure B.5: Generalization performance of ATC-based losses and data augmentation on 173 testing tasks. The tasks on the x-axis are arranged by similarity to training tasks: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. Removing data augmentation from ATC (ATC+Aug) decreases its performance compared to No-Aux representation.

B.2.1 Complexity Reduction: A Closer Look

The unnormalized values of complexity reduction is shown in Figure B.6. At first glance, we see that as complexity reduction increases, generalization performance improves. However, if we look at the values of complexity reduction that are highly close to 0, we see that performance start to degrade. This demonstrates that while having a high level of complexity reduction is desirable, having excessively high levels of complexity reduction is not.

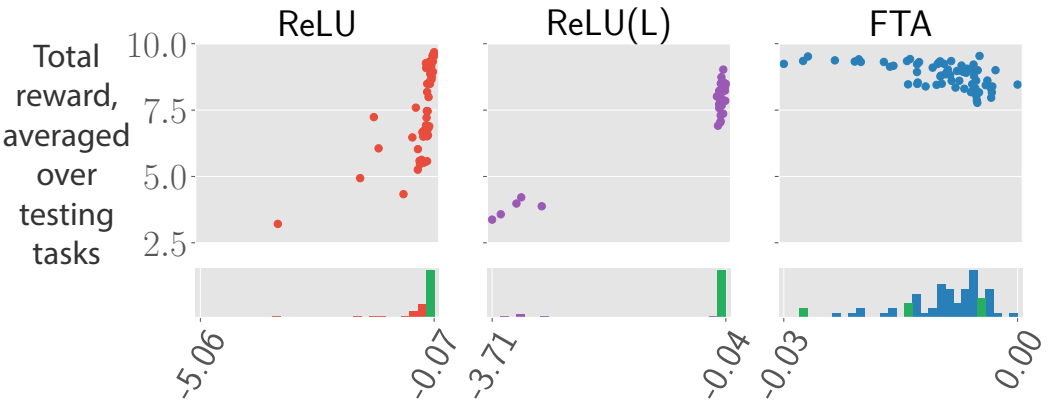

Figure B.6: Generalization performance averaged over testing tasks versus complexity reduction values. Each dot in a plot corresponds to one representation, at the (x,y) point corresponding to its complexity reduction value and average generalization performance. The histograms under the plots visualize the density of points at each property value. Except for extremely high complexity reduction values, generalisation performance improves as complexity reduction grows.

B.2.2 An Explanation for Poor Generalization Performance of the Laplacian Representation

The representation properties of Laplacian are shown in Figure B.7. For ReLU-based representations, the Laplacian task improves dynamics awareness, orthogonality, and sparsity, but it significantly reduces diversity and complexity reduction. This significant reduction in these two measures explains the poor generalization performance of ReLU-based Laplacian representations. For the case of FTA, the this task greatly increases orthogonality, sparsity, and diversity, while it noticeably decreases dynamics awareness. The decrease in dynamics awareness explains the reduction in the performance of FTA-based Laplacian representations.

B.2.3 An Explanation for the Importance of Data Augmentation for ATC

The representation properties of ATC with (labeled ATC) and without (labeled ATC-Aug) data augmentation are shown in Figure B.8. In the case of ReLU activation, the removal of the data augmentation technique results

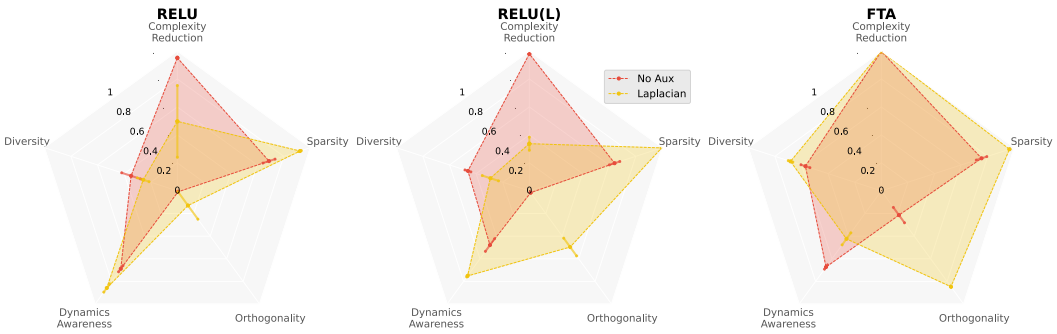

Figure B.7: Properties of LAPLACIAN representations. The LAPLACIAN task damages the generalization performance of ReLU-based representations and slightly decreases the performance of FTA-based representations. Each subplot shows a property value achieved by six different representations: ReLU, ReLU(L), and FTA with LAPLACIAN and ReLU, ReLU(L), and FTA with no auxiliary tasks. It is clear this auxiliary task changes the properties of the representations; particularly dynamics awareness and orthogonality. We did not include non-interference as LAPLACIAN had no impact on it.

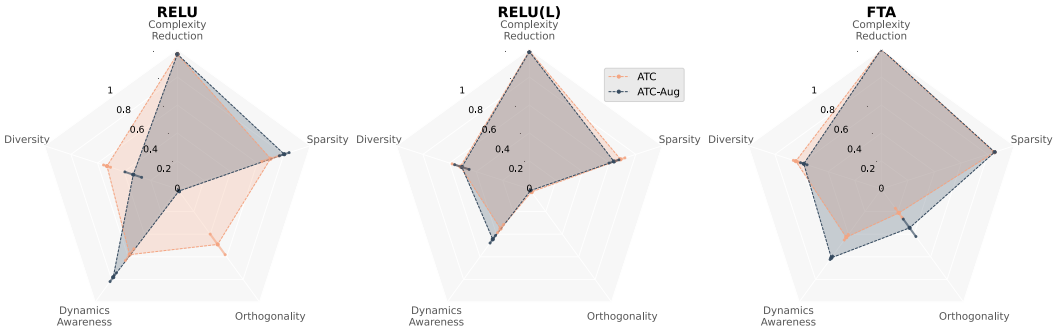

Figure B.8: Comparing properties of ATC representations with and without data augmentation. The ATC task produces FTA, ReLU, and ReLU(L) representations that generalize well, but removing data augmentation (ATC-Aug) results in ReLU representations that generalize poorly. Each subplot shows a property value achieved by six different representations: ReLU, ReLU(L), and FTA with ATC and ReLU, ReLU(L), and FTA with ATC-Aug. We did not include non-interference as both these losses had no impact on it.

in a catastrophic reduction in orthogonality, pushing it to the value of 0. It also reduces diversity and increases the dynamics awareness. The catastrophic decrease in orthogonality, in the case of ReLU, explains why the generalization performance is greatly reduced as we remove the data augmentation from the ATC task. Interestingly, the representation properties are similar in the

case of ReLU(L) and FTA, with a slight increase in dynamics awareness and orthogonality, and a slight decrease in diversity after the removal of data augmentation. This explains why the generalization performance in these two cases has not been affected by the removal of the data augmentation technique.

B.2.4 Relationship between Representation Properties

The relationship between representation properties is shown in Figure B.9. Here, we focus on two subplots that are highlighted with orange color. Diversity and complexity reduction show strong positive linear correlation. This suggest that maximizing one of these measures using an auxiliary loss results in the maximization of the other. This can be clearly seen in Figure 6.4 in Chapter6. As we increase the value of orthogonality linearly, diversity tends to rise logarithmically. However, several outliers imply that this connection between diversity and orthogonality may not always hold.

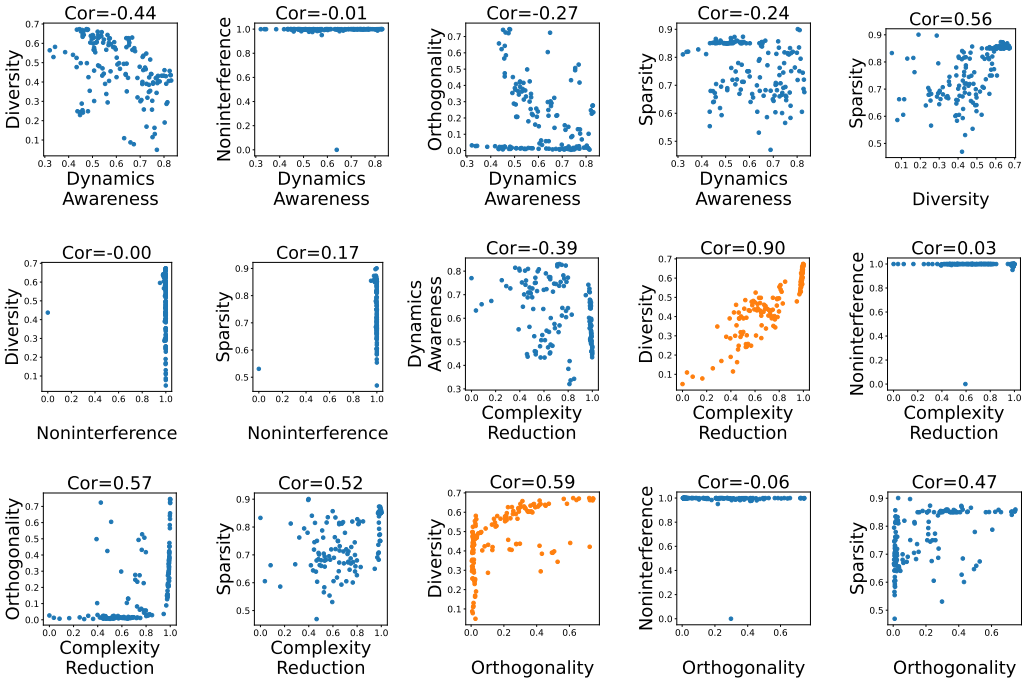

Figure B.9: The correlation between representation properties. There exists strong positive correlation between diversity and complexity reduction. Meanwhile, when diversity is high, the increment of diversity comes with an increment of orthogonality. The above two subplots are highlighted with orange color. Each subplot shows the relationship between a pair of properties. Each dot refers to one representation. The x and y coordinates are property measures. The properties' names are shown as labels on each axis. The correlations (Cor) are reported above each subplot.