Emergent Representations in Reinforcement Learning and Their Properties

by

Han Wang

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

This dissertation investigates the properties of representations learned by modern deep reinforcement learning systems. Representation learning plays an important roll in reinforcement learning. A representation contains information extracted from states—the description of the current situation given by the environment. Therefore, a high-quality representation is not only essential to build a robust reinforcement learning agent but also can help improving learning efficiency. Many sub-problems of reinforcement learning, such as planning with model and directed exploration, can be solved more efficiently with a successful agent state discovery. There are a lot of representation learning algorithms that have been proposed. Much of the earlier work in representation learning for reinforcement learning focused on designing fixed-basis architectures to achieve desirable properties, such as orthogonality. 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—desired representations will emerge under appropriate training schemes. In this work, we discuss how emergent representations learned with different tasks settings, both with and without auxiliary tasks, perform on properties that people think a good representation has. This thesis (1) empirically investigates how these emergent representations relate to historical notions of good representations, and (2) provides novel insights regarding end-to-end training, the auxiliary task effect, and the utility of successor-feature targets. In particular, we will compare the

representations learned by several standard architectures by discussing seven representational properties and studying how these properties relate to control and transfer performance.

Preface

This thesis is based on a pending publication "Emergent Representations in Reinforcement Learning and Their Properties". It is a joint work with Muhammad Zaheer, Raksha Kumaraswamy, Vincent Liu, Adam White and Martha White. Zaheer and I are responsible for the experiments. Martha, Adam and Raksha wrote and edited the main part of the paper, Raksha, Zaheer and I wrote the appendix.

To my family

For all the support and love

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

– Alan Turing

Acknowledgements

I would like to express my sincere appreciation to Dr. Martha White and Dr. Adam White. They have spent long time on discussing the project with me, giving valuable suggestions, helping me with my writing skill, as well as encouraging me. They have taught me a lot not only on solving the problem I met in the project, but also, more importantly, on how to do good research. I would like to thank Dr. Rich Sutton for letting me know there is such a beautiful and elegant subject to study and there are more interesting things to do. I would also like to thank Raksha Kumaraswamy, Muhammad Zaheer, and Vincent Liu, for all the fantastic suggestions and advice, and supporting me during all the difficult time. They are great teammates to work with. Thanks Dylan Ashley, Andrew Jacobsen, and Fei Wang for spending a lot of time on helping me with making this thesis better. I also appreciate Khurram Javed, Yangchen Pan, Banafsheh Rafiee, Sina Ghiassian, Ehsan Imani, Sungsu Lim, Niko Yasui, Chen Ma, Yi Wan and all others in RLAI and AMII, for all their help and support during the past few years. It is a great honor and pleasure to work in this friendly group.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Good Representations for Reinforcement Learning

Reinforcement learning is a subfield in Artificial Intelligence. There are two main components in reinforcement learning: the *agent* which learns through the interaction with the environment to perform a better decision making, and the *environment*, which is defined as other parts outside of the agent. The agent learns through trial-and-error interactions with the environment to choose the optimal action for a given situation that fulfills a specific goal. The degree of success for each action is measured by the *reward* signal, a number provided by the environment. Through estimating the total reward in future if the agent starts from this state (and action), the agent learns a *value function* to determine how good a state (and action) is [66]. An *environment state* or *observation* is a summary of the current situation provided by the environment. The given observation may include noise or details which are useless for solving the task. The agent can learn its own summary of the situation—a *representation*—which critically determines the agent's ability to generalize and discriminate. This step is called *representation learning*, or agent state discovery.

Ultimately, many subproblems in reinforcement learning depend on successful agent state discovery. For instance, in planning, the agent uses a learned model to predict the next agent state, and to generate simulated interactions [61], [62], [68]. In this case, accurately predicting the next state

will help the agent perform the task better [23], [71], [75]. Another example is transfer learning, when the representation is learned in one task and is later used in a different task; the representation may encode shared underlying factors of both tasks, thus reduce the learning time when the task changes [6]. Recent theoretical results [12], [32], [55], [74] highlight the importance of the representation for sample efficient reinforcement learning—that is, a sufficient representation is critical for agent to reach a certain level of performance with a limited amount of data. Simple general principles like value functions and optimism in the face of uncertainty might be all the agent needs to learn a desirable policy, if only we could learn a good representation. The key question is: what is a good representation?

To date, representation learning still remains one of the central challenges in reinforcement learning. Much of the earlier work in representation learning for reinforcement learning focused on designing fixed basis architectures to achieve different properties. Many of these properties are common to the general machine learning setting. Many approaches either use or search for orthogonal or decorrelated features, such as with orthogonal matching pursuit [45], Bellman-error basis functions [47], Fourier basis [29], and tile coding [65], [66]. Prototypical input matching methods have been extensively explored, as in kernel methods, radial basis functions [66], cascade correlation networks [13] and Kanerva coding [26]. Whereas, ideas from nonlinear dimensionality reduction, such as in proto-value functions [28], [37], are not as popular as in supervised learning. Finally, massive expansion architectures, such as random representations [69], tile coding, and sparse distributed memories [52], use sparse connections and activations to reduce computation and increase scalability. The representation learning algorithm that improves one property through a designed loss function or constraint undoubtedly improves the quality of representation respecting the property that is focused on. However, it remains unclear that these properties improve efficiency respect to main task and transfer performance.

Recent developments in representation learning explore a different perspective: we should avoid optimizing specific properties and let the training

data through gradient descent 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 [8], [18], [72], auxiliary losses (i.e., auto-encoding, next observation prediction, and pixel control) [5], [24], and training on a distribution of problems (ala meta-learning [15], [25], [43], [59], [60]). The basic idea underlying all these approaches is that good representations will emerge if the problem setting is complex enough; where a good representation is defined by success on some held-out-test tasks.

One of the commonly used end-to-end representation learning algorithms is Deep Q-Networks (DQN) [22], [34], [40]. As an approach combining reinforcement learning and deep neural network, DQN uses *function approximation* to approximate the value function of state-action pairs (*action values*). By projecting the observation space on hidden layers of the neural network, each hidden layer learns a representation space of the observation. The representation is learned online as the weights in the neural network is updated when the agent interacts with the environment [39]. This algorithm offers a scalable representation learning method, that has been applied to numerous tasks includes playing atari games [40] and robotics [50]. DQN can also be incorporated with model-based reinforcement learning for model learning and planning on the representation space [17] and value function learning [23].

Auxiliary tasks can be added to change the architecture and the complexity of problem setting. Through splitting the last layer of the neural network into multiple heads, multiple tasks can be assigned to different heads then solved together by the same network. The goal of adding auxiliary tasks is to learn a better representation respect to the problem to be solved. Some auxiliary tasks which are related to the main task have been empirically shown to be helpful in several experiments, such as learning an extra policy to maximally change pixels in the observation or maximally activate each bit in the representation [24].

There are many different ways to evaluate and understand these emergent representations. Recent work has explored this question in roughly two ways: what good representations look like, and what capabilities good and bad repre-

sentations allow. The most common approach is to visualize the learned representations [5], [11], [16], [17], [19], [20], [31], [40], [54], [62], [64], [77], [81]. This approach has been used to provide evidence for the emergence of abstraction and compositionally in supervised learning [6], [42], [80]. However, in reinforcement learning the effects of delayed consequences and temporally correlated data makes it difficult to import analysis techniques from other fields, and recent evidence has highlighted that popular approaches like saliency maps may not be totally appropriate [3]. More directly, good representations are previously defined as those that facilitate: efficient learning in complex tasks [24], [77], good performance in new or future tasks [4], [15], [22], [41], [49], [56], [73], [82], learning a model and planning with agent state [17], [20], [31], [60], [62], [79], and discovering structure and understanding the world the way humans do [16], [21]. Though there have been work defining properties that a representation should have and work discussing how representations affect agents' performance, there lack of discussion on how those properties related to the performance. Furthermore, it is not always possible to evaluate the representation by looking at the performance, because the cost of running tasks can be large. Thus, we need new methods to describe and evaluate the representation before actually executing the task. Looking into the relationship between the representation's properties and its learning performance will be helpful for us to describe and understand a learned representation.

1.2 Contribution

This work is an exploratory study. We explore the properties of representations learned by modern deep reinforcement learning systems, In addition to visualization and different performance criteria, it can also be beneficial to understand the properties of learned representations—particularly as they are now implicit in the training regime. Specific properties can still be induced by specific optimization techniques and training curriculum.

This thesis investigate the properties of representations that emerge, under standard reinforcement learning training regimes, more specifically,

1. we propose a set of representation properties based on classic notions of good representation and practical approaches to measure them,

2. we provide a methodology to evaluate emergent representations, that should help future investigations,

3. we conduct a thorough empirical study, in two designed environments, and provide novel insights about properties of learned representations, and relationships to transfer performance. We mainly focus on capacity (complexity reduction, dynamics-awareness, linear probing accuracy, and diversity), independence (orthogonality and decorrelation), and robustness (non-interference) of the representation,

4. we provide novel insights regarding end-to-end training and the auxiliary task effect by measuring the list of properties proposed and looking at the transfer performance.

1.3 Thesis Structure

The next chapter describes the mathematical framework for formulating the interaction between the agent and the environment. It also includes the notation used in this thesis. Next, the specific properties we measure and the rational behind why we investigate these exact properties is explained in Chapter 3. Then, Chapter 4 talks about the neural network architecture we use for learning representations. The experiment details including environment description and parameter settings, results, and discussion, are included in Chapter 5. The last chapter (Chapter 6) serves as a conclusion for the work and discusses possible directions for future work.

Chapter 2

Background

This chapter briefly describes the problem fomulation and the algorithms related to our experiments.

2.1 Finite Markov Decision Process

The problem is formalized as a Markov Decision Process (MDP), a framework modelling a discrete-time process. The MDP is a 4-tuple including state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $P : \mathcal{S} \times \mathcal{A} \times S \to [0, \infty]$, and reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. In our case, we focus on finite state space and finite action space (finite MDP). On time step t, the agent is in state S_t, takes action A_t, transits to state $S_{t+1} \sim P(\cdot|S_t, A_t)$ and receives reward R_{t+1}. The discount function $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ specifies a horizon, where the generalization beyond constant γ allows for more general problem specification [67], [78]. The agent's objective is typically to find a policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes the expected discounted sum of rewards – the *return*, $G_t = \sum_{\infty}^{i=0} \gamma^i R_{t+i+1}$.

2.2 Value Functions

The goal of a reinforcement learning agent is generally defined by the reward signal, a number given by the environment. The agent learns to maximize the total reward it obtains in the long run. To reach this goal, an agent needs to estimate how good a state is – that is to estimate the return it can obtain if starting from this state. The value function gives the above estimation, a

state with a higher value is defined as a better state.

The *state value function* $V(s)$ estimates the expected return given a state s. Taking the expected value, it can be denoted as

$$v_\pi(s) \doteq \mathbf{E}_\pi[G_t|S_t = s].$$

By maximizing the value over all states in the state space, we get the *optimal value function*

$$v_*(s) \doteq \max_\pi v_\pi(s).$$

To estimate how good an action is, given the current state, the *action-value function* is applied. Different from the state value function, the action-value function takes both the state and action as input. The expected and optimal action-value functions are:

$$q_\pi(s, a) \doteq \mathbf{E}_\pi[G_t|S_t = s, A_t = a]$$

and

$$q_*(s) \doteq \max_\pi q_\pi(s, a).$$

The policy used in the optimal value function π_* is called an *optimal policy*. Though there can be more than one optimal policy, they share the same value function.

2.3 Function Approximation and Q-learning

The policy is chosen according to the learned value function. Thus we need an algorithm to learn the value estimation through interactions with the environment. Q-learning [76] is often used in this case. It iteratively updates the action-value to estimate $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, with parameters $\theta \in \mathbb{R}^d$, towards the goal of approximating the optimal action-value $Q_* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Q_* is defined as the action-values that satisfy the Bellman optimality equation.

$$Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma_{t+1} \max_{a' \in \mathcal{A}} Q^*(S_{t+1}, a')|S_t = s, A_t = a]$$

It is updated iteratively with

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

where $R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ is the *temporal difference error* (TD-error) and α refers to the learning rate that controls the learning speed of the agent.

In small domains, the true action-values of all states can be mathematically solved. However, as the state space increasing, the computational resource increases fast, thus determining the exact true values mathematically becomes more difficult. To solve the problem with limited computational resources, one can apply the function approximation to learning the action values. With this method, we learn a function parameterized by θ, taking the state and action as inputs then giving the estimated action-value. The neural network is a commonly used function approximator.

When using neural networks for Q_θ, it is common to augment Q-learning with target networks and mini-batch updating from an experience replay buffer, as in Deep Q-Networks (DQN) [40]. A frequent change of the target causes unstable learning in reinforcement learning. Both the replay buffer and the target network in DQN helps with improving the stability. By saving old transitions in a buffer and randomly sample a batch at each step, the buffer makes a dataset where the elements of the dataset are less temporally correlated. The target network contributes to the stable learning by providing a target network Q_{θ^-} which is a copy of Q_θ but is updated slower. It is synchronized with the behaviour network every k steps, while Q_θ is updated every step. When calculating the TD error, DQN takes $r_{i+1} + \gamma_{i+1} \max_{a' \in \mathcal{A}} Q_{\theta^-}(s_{i+1}, a') - Q_\theta(s_i, a_i)$ as the TD-error, where i refers to the index of a randomly sampled transition from the buffer.

2.4 Neural Network Specification

In this work, for the simplicity of explanation, when we describe the network used in DQN, we separate it into 2 parts. The first part is used for learning the representation, the second part takes the learned representation as input then learns the value function. When there exists an auxiliary task, another head is added after the first part for learning the auxiliary task. The architecture

combining DQN with auxiliary tasks is shown in Figure 2.1. More details on the auxiliary tasks are given in Chapter 4 and Chapter 5.

The representation learned by the neural network is typically improved using auxiliary tasks, as shown in Figure 2.1. The first layers, parameterized by θ_R, produce representation $\phi_{\theta_R}(s)$. The last layers uses that representation, with parameters θ_M, to estimate the action-values. Except for the representation layer and the output layer, the rest of the neural network uses activation functions to introduce the nonlinearity. For example, with Rectified Linear Unit (ReLU, the equation is shown in Eq 2.1) as the activation function, $\phi_{\theta_R}(s)$ could have 2 larger ReLU layers followed by a compact (bottleneck) layer: $\phi_{\theta_R}(s) = \text{ReLU}(\text{ReLU}(sW_1)W_2)W_3$, where $s \in \mathbb{R}^k$, $W_1 \in \mathbb{R}^{k\times 128}$, $W_2 \in \mathbb{R}^{128\times 128}$, $W_3 \in \mathbb{R}^{128\times 32}$, $\theta_R = \{W_1, W_2, W_3\}$. The estimate for $Q_\theta(s, a)$ with θ_M could involve a projection back up to a larger feature space of 128, followed by a linear prediction: $Q_\theta(s, a) = \text{ReLU}(\phi_{\theta_R}(s)W_4)W_5$ where $W_4 \in \mathbb{R}^{32\times 128}$, $W_5 \in \mathbb{R}^{128\times 1}$ and θ_M includes $\{W_4, W_5\}$ of all actions. This distinction of making the bottleneck layer the representation is arguably arbitrary, until we add auxiliary tasks to force feature re-use. For example, to further constrain $\phi_{\theta_R}(s)$, we can add next state prediction as an auxiliary task, with parameters θ_A. Now $\phi_{\theta_R}(s)$ needs to adjust to both be useful to predict action-values and next state.

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

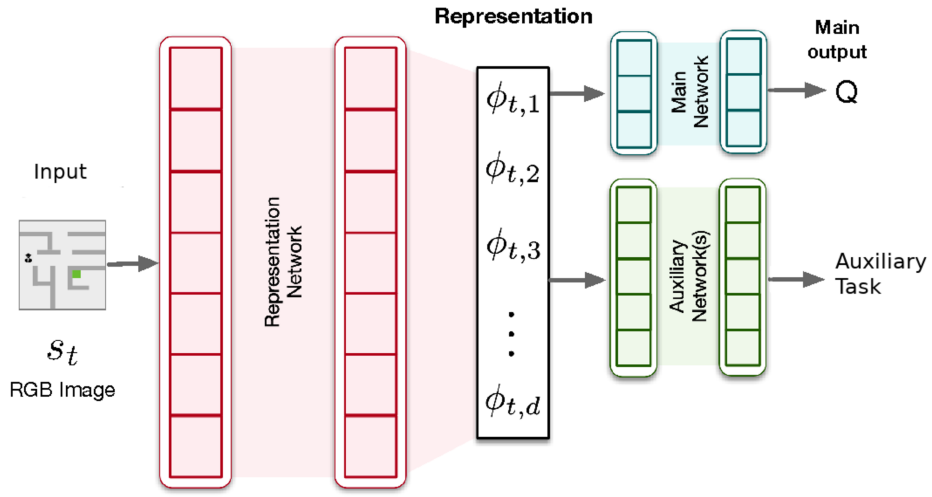

Figure 2.1: We experiment with agents using this network architecture, with different auxiliary losses. The representation network, ϕ learns a mapping from input-state s_t to the agent-state (representation of s_t). This ϕ is learned to improve two objectives: performance on a main task and on an auxiliary task. The diagram depicts the auxiliary tasks we use in this work, described in Chapter 4. Our agent only uses one type of auxiliary task, though of course it could choose to use multiple combinations.

Chapter 3

Representational Properties

Before the emergence of deep reinforcement learning, the study of representations and their effects on learning was focused on a fixed basis. The problem with this is not that a fixed basis cannot capture complex non-linear relationships, but rather that the representation is fixed—the features do not adapt to the problem. In some sense, this is a good thing because it forces the agent designer to consider what are desirable representation properties—a level of analysis complementary to the design of good algorithms. Over the years researchers proposed and debated numerous properties [6], [33], [34]. In this work, we measure 7 of them for analyzing the representations' performance. Table 3.1 provides a summary of these measures, and more detailed review can be found in the following sections.

3.1 Capacity

The first important property to consider for representation is the *capacity*: can it represent the functions we want to learn? The assumption is that if the representation has a high capacity, the value function network can be a simple function of these features, such as linear functions or simple neural networks. We consider 4 properties reflecting the capacity, one direct measure—complexity reduction (Section 3.1.1), and three indirect measures—dynamics-awareness (Section 3.1.2), linear probing accuracy (Section 3.1.3), and diversity (Section 3.1.4.

Table 3.1: Representation Properties. All measures are normalized, to be between $[0, 1]$. A value of 0 means a representations does not have that property, and 1 is that it has the property maximally. The feature vectors are computed on a set of N samples, to get $\{\phi_1, ..., \phi_N\}$, with $Q_i \doteq \max_a Q(s_i, a)$.

Name	**Measure**	**Description**
Complexity Reduction	$1 - L_{\text{rep}} / \max_{x \in \{rep\}} L_x$ $L_{\text{rep}} \stackrel{\text{def}}{=} \max_{i,j:\phi_i \neq \phi_j} \frac{\lvert Q_i - Q_j \rvert}{\lVert \phi_i - \phi_j \rVert}$	Learning a value function on a representation space with a lower Lipschitz constant is easier. Normalization is done relative to other representations, normalizing by the largest L.
Linear Probing Accuracy	1-$\frac{1}{N} \sum_{i=1}^{N} \frac{\lvert \theta_y \phi_i - y_i \rvert}{\lvert y_i \rvert + 1}$, (regression) $\frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\arg\max \theta_y \phi_i = y_i)$, (classification)	Representation can predict expert-defined variables useful for the task, as percentage accuracy. θ_y is learned using least-squares loss for regression, and cross-entropy for classification.
Dynamics Awareness	$\frac{\sum_i^N \lvert\lvert \phi_i - \phi_{j \sim U(1,N)} \rvert\rvert - \sum_i^N \lvert\lvert \phi_i - \phi_i' \rvert\rvert}{\sum_i^N \lvert\lvert \phi_i - \phi_{j \sim U(1,N)} \rvert\rvert}$	Nearby states should have similar representations, far apart state dissimilar representations.
Diversity ($1 -$ *Specialization*)	$1 - \frac{\sum_{i,j,i<j} (\delta_{s,i,j} - \bar{\delta_s})(\delta_{q,i,j} - \bar{\delta_q})}{\sqrt{\sum_{i,j,i<j} (\delta_{s,i,j} - \bar{\delta_s})^2 \sum_{i,j,i<j} (\delta_{q,i,j} - \bar{\delta_q})^2}}$, $\delta_{s,i,j} \stackrel{\text{def}}{=} \lVert \phi_i - \phi_j \rVert$, $\delta_{q,i,j} \stackrel{\text{def}}{=} \lvert \max_a (Q(\phi_i, a) - Q(\phi_j, a)) \rvert$	Reflects that this representation is tailored to this specific Q.
Orthogonality	$1 - \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{\lvert \langle \phi_i, \phi_j \rangle \rvert}{\lVert \phi_i \rVert_2 \lVert \phi_j \rVert_2}$	Feature vectors for different states should be maximally different.
Decorrelation	$1 - \frac{2}{d(d-1)} \sum_{k,j,k<j}^{d} \lvert \text{corr}(\mathbf{v}^k, \mathbf{v}^j) \rvert$ $\text{corr}(\mathbf{v}^k, \mathbf{v}^j) = \frac{\sum_{i=1}^{N} \mathbf{v}_i^k \mathbf{v}_i^j}{\lVert \mathbf{v}^k \rVert_2 \lVert \mathbf{v}^j \rVert_2}$ $v_i^k \stackrel{\text{def}}{=} \phi_i^k - \bar{\phi^k}$, $\mathbf{v}^k \stackrel{\text{def}}{=} [v_1^k, \ldots, v_N^k]^T$	Features are as decorrelated as possible.
Non-interference	$1 - \frac{2}{N(N-1)}$ $\sum_{i,j,i<j}^{N} \max\left(0, -1 \times \frac{\langle g_i, g_j \rangle}{\lVert g_i \rVert_2 \lVert g_j \rVert_2}\right)$, $g_i = \frac{\partial L(\phi_i)}{\partial \theta_M}$ $L(\phi_i) =$ $\frac{1}{2}(r_i + \gamma_i \max_{a'} Q(\phi_i', a') - Q(\phi_i, a_i))^2$	Update in one state points in the same direction as the update in another state, on average.

3.1.1 Complexity Reduction

For *complexity reduction*, we directly measure the complexity of the value function learned with a representation.

We measure the complexity by approximating the Lipschitz constant for the value functions. The measured constant is normalized by the maximum value.

$$\text{Complexity Reduction} = 1 - \frac{L_{\text{rep}}}{\max_{x \in \{rep\}} L_x}$$
$$L_{\text{rep}} \stackrel{\text{def}}{=} \max_{i,j:\phi_i \neq \phi_j} \frac{|Q_i - Q_j|}{\|\phi_i - \phi_j\|} \tag{3.1}$$

In the above measure, i and j are indices of environment states from our dataset, ϕ_i and ϕ_j are the representation of states s_i and s_j separately, and Q_i and Q_j are their action values respecting the action which causes the maximal action value difference.

A small complexity score means features encoded much of the non-linearity needed. On the contrary, a larger complexity implies that the value function needs to be more complex to learn a reasonable policy, thus, is harder to learn. This means lower complexity is a desired property for representations.

3.1.2 Dynamics-awareness

Along the same lines, we can also indirectly measure complexity without specifying a set of value functions—by testing if the representation is *dynamics-aware*. 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 a low similarity. This measure is related to the Laplacian used for proto-value functions and successor features [36], [63]. This property is measured using Formula 3.2, where ϕ'_i represents the next agent state of the i_{th} sample. The intuition behind this measure is that the distance between representations of two consecutive states should be less than the distance between representations of two random states. The consecutive states pairs are s_i and the next state s'_i, we also uniformly sample another state s_j to form a random state pair with s_i.

$$\text{Dynamics} = \frac{\sum_i^N ||\phi_i - \phi_{j\sim U(1,N)}|| - \sum_i^N ||\phi_i - \phi_i'||}{\sum_i^N ||\phi_i - \phi_{j\sim U(1,N)}||} \tag{3.2}$$

3.1.3 Linear Probing Accuracy

The second indirect measure is to predict a set of expert-defined factors that the agent would likely benefit from. We assume that to solve the task, the agent must know the key information, in other words, the representation should contain the key information extracted from the observation. For example, in a top-down image in a navigation grid world, the agent would likely benefit from retaining its (x, y) position in the representation and be able to predict it. We test if these expert factors can be linearly predicted by the representation (*linear probing - xy*, *linear probing - count*, and *linear probing - color*). This approach was introduced as linear probing for classification [1], and later used in reinforcement learning [2]; therefore we call it *linear probing accuracy*. When the factor predicted by the linear transformation is formed as a regression problem, such as predicting the coordinate of the agent, we report the negative value of percentage error added by one. In the case of a classification problem, like predicting the color of an object in the environment, we use the accuracy instead (Formula 3.3).

$$\begin{aligned} \text{Regression Accuracy} &= 1 - \frac{1}{N}\sum_{i=1}^{N} \frac{|\theta_y \phi_i - y_i|}{|y_i| + 1} \\ \text{Classification Accuracy} &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\arg\max \theta_y \phi_i = y_i) \end{aligned} \tag{3.3}$$

where N is the number of samples in the dataset, θ_y refers to weights in the linear transformation system. Note that we add one on the denominator to avoid dividing by 0. In classification case, the $\arg\max$ means to choose the class with the highest probability.

3.1.4 Diversity and Specialization

Finally, we can measure how much a representation has *specialized* to a particular value function. If the representation is specialized to a small sub-

set of all possible value functions, it may not be as useful for learning other functions. This property allows us to further indirectly measure representation capacity, as we are able to check the level of specialization for a given function without needing to have access to the larger set of possible functions. If the representation is specialized, then we might expect to see a tight correspondence between distances in representations and values. For two input states s_i and s_j, if $\delta_{s,i,j} = \|\phi_{\theta_R}(s_i) - \phi_{\theta_R}(s_j)\|$ is small (big) then $\delta_{q,i,j} = |\max_a Q_{\theta_M}(s_i, a) - \max_a Q_{\theta_M}(s_j, a)|$ should be small (big), and vice versa. This can be measured by using the correlation between $\delta_{s,i,j}$ and $\delta_{q,i,j}$ for across s_i, s_j. We report *diversity* = 1 - *specialization*, as a potentially useful property to have.

$$\begin{aligned} \text{Diversity} &= 1 - \text{Specialization} \\ \text{Specialization} &= \frac{\sum_{i,j,i<j}(\delta_{s,i,j} - \bar{\delta_s})(\delta_{q,i,j} - \bar{\delta_q})}{\sqrt{\sum_{i,j,i<j}(\delta_{s,i,j} - \bar{\delta_s})^2 \sum_{i,j,i<j}(\delta_{q,i,j} - \bar{\delta_q})^2}} \\ \delta_{s,i,j} &\stackrel{\text{def}}{=} \|\phi_i - \phi_j\| \\ \delta_{q,i,j} &\stackrel{\text{def}}{=} |Q_i - Q_j| \end{aligned} \tag{3.4}$$

3.2 Independence

Only performing well in the capacity may not be enough to learn on transfer tasks, thus we also consider other functional properties of the feature such as reducing redundancy in the representation—finding *linearly independent* features. This property can be measured through orthogonality (Section 3.2.1) and decorrelation (Section 3.2.2).

3.2.1 Orthogonality

Orthogonal representations satisfy the redundancy reduction requirement, and additionally provide distributed features as well as minimal interference, for example, a dense set of orthogonal (latent) factors found by factor analysis. A representation like this is highly distributed, as the learned factors act as a basis of the space, each input can be represented by more than one feature. At the same time, orthogonality reduces the interference: the interference for

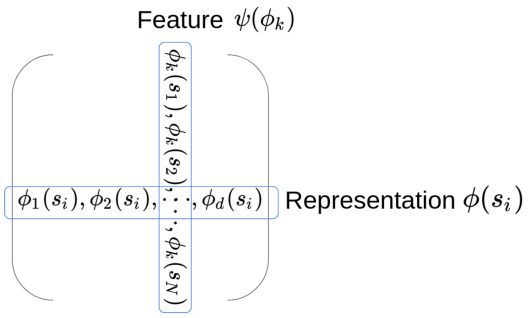

Figure 3.1: Writing representations of all samples as a matrix, then rows are representations and columns are features.

two states with orthogonal feature vectors is zero under linear updating. If two states s_1 and s_2 have $\phi(s_1)^\top \phi(s_2) = 0$, then performing an update in s_1, $\tilde{w} = w + \alpha\delta\phi(s_1)$ has no impact on the prediction in s_2: $\phi(s_2)^\top \tilde{w} = \phi(s_2)^\top (w + \alpha\delta\phi(s_1)) = \phi(s_2)^\top w + \alpha\delta\phi(s_2)^\top \phi(s_1) = \phi(s_2)^\top w$. We measure the cosine similarity between representations. If the cosine similarity is zero, then they are orthogonal to each other. The absolute value is taken to prevent that the negative value is cancelled out by the positive value.

$$\text{Orthogonality} = 1 - \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{|\langle \phi_i, \phi_j \rangle|}{\|\phi_i\|_2 \|\phi_j\|_2} \tag{3.5}$$

Relationship between Orthogonal Representations and Orthogonal Features

Notice that the *representation* refers to the learned vector of each state, and we use the *feature* to describe each dimension of the representation. In this section, we show that checking the orthogonality between representations is equivalent to checking the orthogonality between features.

Assume we have an finite set of input-states $\{s_1, s_2, \ldots, s_N\}$ represented in a d-dimensional space, where the corresponding representations are $\phi(s_i) = [\phi_1(s_i), \phi_2(s_i), \ldots, \phi_d(s_i)]^\top$ (ϕ_k being a function to produce the k-th feature dimension), and $\psi(\phi_k) = [\phi_k(s_1), \phi_k(s_2), \ldots, \phi_k(s_N)]^\top$. More clearly, if the representations of all states are written in rows as a matrix, then each column is a feature vector, as shown in Figure 3.1.

With this, below we show that $\sum_{i,j}^{n}(\phi(s_i)^\top\phi(s_j))^2 = \sum_{k,l}^{d}(\psi(\phi_k)^\top\psi(\phi_l))^2$.

$$
\begin{aligned}
\sum_{i,j}^{n}(\phi(s_i)^\top\phi(s_j))^2 &= \sum_{i,j}^{n}(\phi(s_i)^\top\phi(s_j))(\phi(s_i)^\top\phi(s_j)) \\
&= \sum_{i,j}^{n}\left(\sum_{k=1}^{d}\phi_k(s_i)\phi_k(s_j)\right)\left(\sum_{l=1}^{d}\phi_l(s_i)\phi_l(s_j)\right) \\
&= \sum_{i,j}^{n}\sum_{k,l}^{d}\phi_k(s_i)\phi_l(s_i)\phi_k(s_j)\phi_l(s_j) \\
&= \sum_{k,l}^{d}\sum_{i,j}^{n}\phi_k(s_i)\phi_l(s_i)\phi_k(s_j)\phi_l(s_j) \\
&= \sum_{k,l}^{d}\left(\sum_{i=1}^{n}\phi_k(s_i)\phi_l(s_i)\right)\left(\sum_{j=1}^{n}\phi_k(s_j)\phi_l(s_j)\right) \\
&= \sum_{k,l}^{d}(\psi(\phi_k)^\top\psi(\phi_l))(\psi(\phi_k)^\top\psi(\phi_l)) \\
&= \sum_{k,l}^{d}(\psi(\phi_k)^\top\psi(\phi_l))^2
\end{aligned}
$$

Therefore, when the sample-space is not enumerable, that is ϕ_k is infinite-dimensional, orthogonality of representations may be used as a surrogate for measuring the orthogonality of features.

3.2.2 Decorrelation

Two properties related to orthogonality are decorrelation and sparsity. If the correlation between pairs of features is small, then the features are *decorrelated*. If only a small number of features are activated for an input, then the features are *sparse*, with an additional condition that each feature is active for some inputs (in other words, no dead features). 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. Besides orthogonality, we also measure decorrelation (Formula 3.6), but omit sparsity as we use a bottleneck architecture and the representation is supposed to be dense. For expansionist architectures, sparsity could be included, as an overlapping but still complementary property to orthogonality and decorrelation.

$$
\begin{aligned}
\text{Decorrelation} &= 1 - \tfrac{2}{d(d-1)} \sum_{k,j,k<j}^{d} |\text{corr}(\mathbf{v}^k, \mathbf{v}^j)| \\
v_i^j &\stackrel{\text{def}}{=} \phi_i^j - \bar{\phi^j} \\
\mathbf{v}^k &\stackrel{\text{def}}{=} [v_1^k, \ldots, v_N^k]^T \\
\text{corr}(\mathbf{v}^k, \mathbf{v}^j) &= \frac{\sum_{i=1}^{N} \mathbf{v}_i^k \mathbf{v}_i^j}{\|\mathbf{v}^k\|_2 \|\mathbf{v}^j\|_2}
\end{aligned}
\tag{3.6}
$$

Relationship between Orthogonal and Uncorrelated features

When the features are centered, decorrelation and orthogonality are equivalent; otherwise, they are not precisely the same [7]. Next, We show the relationship between orthogonal features and uncorrelated features.

Assume we have an finite set of input-states $\{s_1, s_2, \ldots, s_n\}$ represented in a d-dimensional space, where the corresponding representations are $\phi(s_i) = [\phi_1(s_i), \phi_2(s_i), \ldots, \phi_d(s_i)]^\top$ (ϕ_l being a function to produce the l-th feature dimension), and $\psi(\phi_i) = [\phi_i(s_1), \phi_i(s_2), \ldots, \phi_i(s_n)]^\top$. Let $\bar{\psi}(f_i) = [\bar{\phi_i}, \bar{\phi_i}, \ldots, \bar{\phi_i}]^\top$ where $\bar{\phi_i} = \frac{1}{n} \sum_{j=1}^{n} [\phi_i(s_j)]$, denoting the expected value of feature i over the set of input-states. If all features are centered, that is, $\bar{\phi_i} = 0$ for all i, then it is trivial to see that

$$
\frac{1}{n^2} \sum_{k,l}^{d} (\psi(\phi_k) - \bar{\psi}(\phi_k))^\top (\psi(\phi_l) - \bar{\psi}(\phi_l)) = \frac{1}{n^2} \sum_{k,l}^{d} \psi(\phi_k)^\top \psi(\phi_l).
$$

The LHS is a measure of correlation and the RHS is a measure of orthogonality.

3.3 Robustness

More recent work [35] in neural networks has also focused on *robustness*. Both interference and noise are related to the robustness of representation. It is common to investigate if representations capture certain *invariances* and whether they are *robust to input-noise*, particularly to adversarial noise [35]. The design of meaningful invariances and noise is non-trivial, and environment-specific, so we omit these properties from this study and focus on interference only.

3.3.1 Non-interference

Interference reflects how much updates in one state reduce accuracy in other states; it can be thought of as bad generalization, that different inputs generate gradients in different or even opposite directions so that the gradient given by the later input cancels out the gradient of the earlier input. It is typically difficult to measure whether the generalization is good or bad across value functions, but we can do so for a specific value function. We use a random initialization of θ_M and then measure gradient alignment [57]. This measure reflects that if gradients point in opposite directions, then interference is likely to occur: $\frac{\langle g_i, g_j \rangle}{\|g_i\|_2 \|g_j\|_2} < 0$, where g_i and g_j are gradients when updating the value functions of the i_{th} and the j_{th} states.

$$\begin{aligned} \text{Non-Interference} &= 1 - \tfrac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \max\left(0, -1 * \frac{\langle g_i, g_j \rangle}{\|g_i\|_2 \|g_j\|_2}\right) \\ g_i &= \frac{\partial L(\phi_i)}{\partial \theta_M} \\ L(\phi_i) &= \frac{1}{2}(r_i + \gamma_i \max_{a'} Q(\phi_i', a') - Q(\phi_i, a_i))^2 \end{aligned} \tag{3.7}$$

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 evaluation like visualization. Such a list is necessarily incomplete; we attempt only to start with a reasonably broad set of properties.

Chapter 4

Representations Learning Architectures

A common approach to induce useful representations is through the use of auxiliary losses. The cross-combination of auxiliary losses with different network architectures represents a vast array of possible agents to study. In this work, our focus is the representations that emerge under different auxiliary losses, with the same fixed network architecture depicted in Figure 2.1. The intuitions are 1) the functional capacity and size of all the learned representations are equivalent, making comparisons more interpretable, and 2) the role and impact of auxiliary losses in reinforcement learning remains poorly understood, and constitutes an important area of study. We consider six auxiliary losses, four of which are based on predictions (Section 4.1) and the last two are based on control (Section 4.2).

Throughout we use environment Simple Maze to make it concrete. Simple Maze is a grid world environment. It uses the RGB image of the current situation as a state. The agent starts randomly from an empty pixel and is trained to go to the goal area at $[9, 9]$. There are 4 actions regarding to 4 directions, each of which moves the agent to the adjacent pixel, but the position of the agent remains the same when the action brings the agent into the wall. Figure 4.1 shows a random state of the Simple Maze. More details are given in Section 5.1.1.

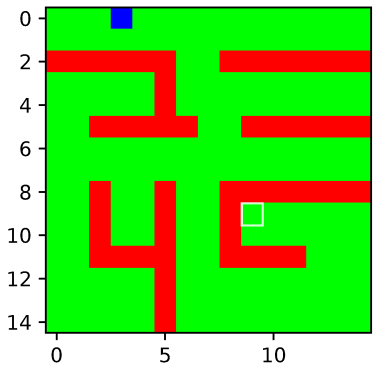

Figure 4.1: A random state of Simple Maze. The position of the agent is shown by a blue pixel. Red blocks are walls and the background is green. The goal area is shown in the white squre. However, it is invisible to the agent. In the plot that the agent gets, the goal state has the same color as the background.

4.1 Prediction Based Auxiliary Tasks

We have four auxiliary tasks learning extracting information from the observation, such as reconstructing the observation from the representation, and extracting the coordinate of the agent. Some tasks predict factors related to the dynamics of the environment, like predicting the next agent state or successors.

4.1.1 Input Decoder

The first auxiliary task we consider is reconstruction of the environment-state, which we call *Input-Decoder*. The goal in such autoencoders is to capture key latent factors. The representation learned by a linear autoencoder has been shown to be equivalent to principal components analysis (PCA) [6]. 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. It has been previously used as an auxiliary task in RL [30], [36]. We do not expect this auxiliary loss to be particularly effective in RL, but we nonetheless include it as a classic and simple choice. The architecture is shown in Figure 4.2.

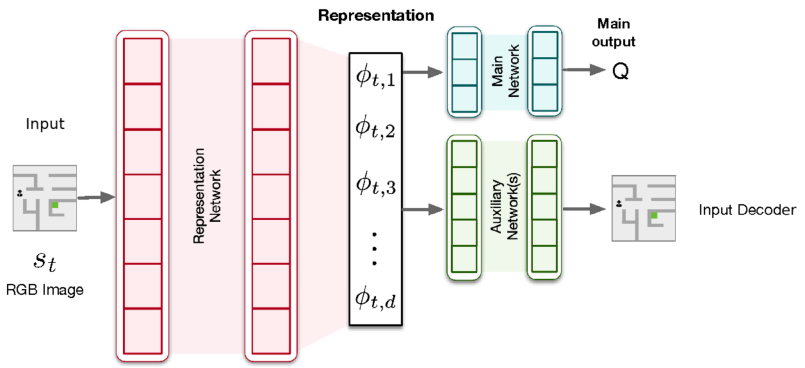

Figure 4.2: The network architecture learning representation with the input decoder auxiliary task. The agent reconstructs the input state in the auxiliary task.

4.1.2 Next Agent State Prediction

Another natural choice is to predict *Next Agent State*, which is a common auxiliary loss [9], [17], [24], [43], [48], [59], [62], [79]. This loss encourages the representation to capture transition dynamics. The agent predicts ϕ_{t+1}, using ϕ_t. This might seem a bit circular, and indeed without the main task, could have vacuous solutions. 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 theoretical papers have highlighted that the ability to predict next state is related to the ability to predict action-values [5], [46], [70]. The architecture is shown in Figure **??**. In practice, we predict the difference between the next and the current agent state ($\delta_t = \phi_{t+1} - \phi_t$), rather than predicting the next state directly. The loss is written as

$$L = \sum_{i=1}^{d} (\hat{\delta_{t,i}} - \delta_{t,i})^2 + \frac{1}{d} \sum_{i=1}^{d} \max(0, 1 - \delta_{t,i})$$

where $\hat{\delta_{t,i}}$ represents the predicted difference and the second part is a repulsive loss to prevent the value to be too small. c

4.1.3 Successor Feature Prediction

The Next Agent State auxiliary loss can be taken one step further, with the target including not just the next agent-state but many future agent states. *Successor Features* provide just such a target. For environment-state s_t, the successor features are defined with respect to a particular policy π as $\psi^\pi(s_t) = \mathbb{E}\big[\sum_{i=1}^{\infty} \gamma^{i-1}\phi(s_{t+i})\big]$, where we use a fixed $\gamma \in [0, 1)$ for simplicity, though general discount functions can also be used. Successor features have been particularly motivated as a way to generalize quickly in reinforcement learning, as value estimates can be quickly inferred for new reward functions that are a linear function of agent-state $\phi(s_t)$ [4], [58]. For tabular features, successor features correspond to successor representations, which have an equivalence to the Laplacian [36].

The predictions for these targets can be estimated online using temporal difference learning. For $\theta_{A,j}$, we can update the value estimate $V_{\theta_{A,j}}(\phi_t) \approx \psi^\pi(s_t)_j$ using $\theta_{A,j} = \theta_{A,j} + \alpha\delta\nabla V_{\theta_{A,j}}(\phi_t)$ where $\delta = \phi_{t+1,j} + \gamma V_{\theta_{A,j}}(\phi_{t+1}) - V_{\theta_{A,j}}(\phi_t)$. With $\gamma = 0$, this auxiliary loss corresponds to predicting next agent state, and so it is a strict generalization. Furthermore, there is another constraint that the reward r_{t+1} should be linearly predictable given ϕ_t, which is done by a linear transformation added as another head. The choice of policy π does make this auxiliary loss more complex than predicting next agent state. We opt to use the greedy policy according to the action-values for the main task, which means the successor features are tracking a changing policy. The architecture is shown in Figure 4.3.

4.1.4 Expert-designed Targets Prediction

The final prediction auxiliary loss is based on *Expert-Designed Targets* used for linear probing. These key variables are likely also useful auxiliary tasks. Though it may not always be possible to design such expert targets, we include it here as a reasonable baseline and sanity check. As an auxiliary task, these targets are predicted with a two layer network under, as in Figure 2.1, rather than linearly. The architecture is shown in Figure 4.4.

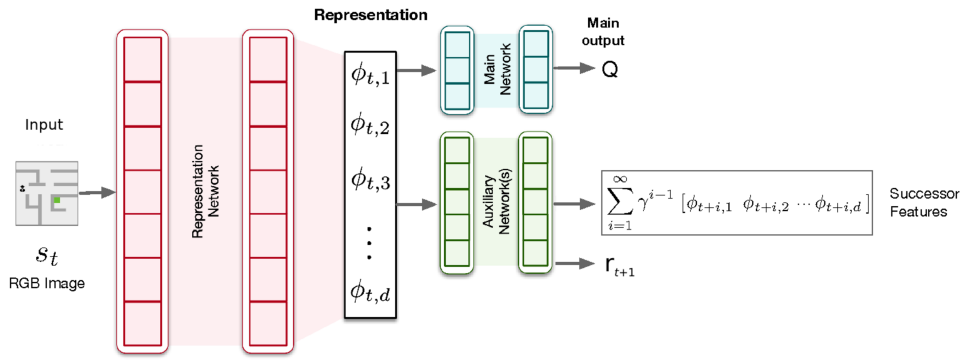

Figure 4.3: The network architecture learning representation with the successor feature prediction auxiliary task. The agent predicts the successor feature of the current agent state in the auxiliary task.

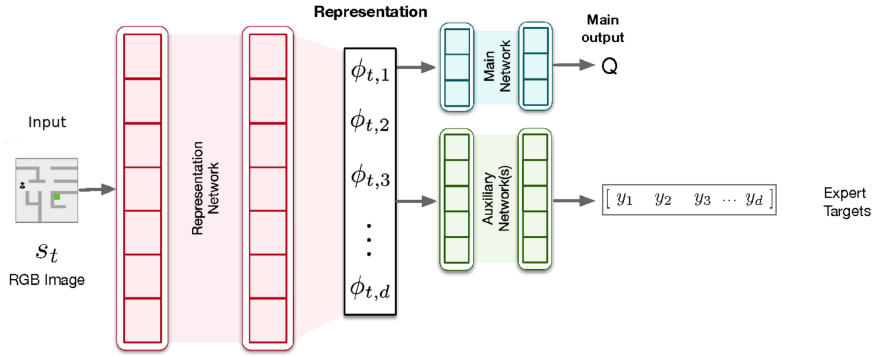

Figure 4.4: The network architecture learning representation with the expert target auxiliary task. The agent predicts the target defined by the expert knowledge in the auxiliary task.

4.2 Control Based Auxiliary Tasks

Next we describe the auxiliary control tasks. The auxiliary control does not affect the terminal and reward in the main task. It is just an extra value function learned with off-policy with a different goal state as in the main task. In the off-policy control, except that the reward function or goal state is changed according to the auxiliary task setting, other dynamics remain the same as in the main task. The architecture is shown in Figure 4.5.

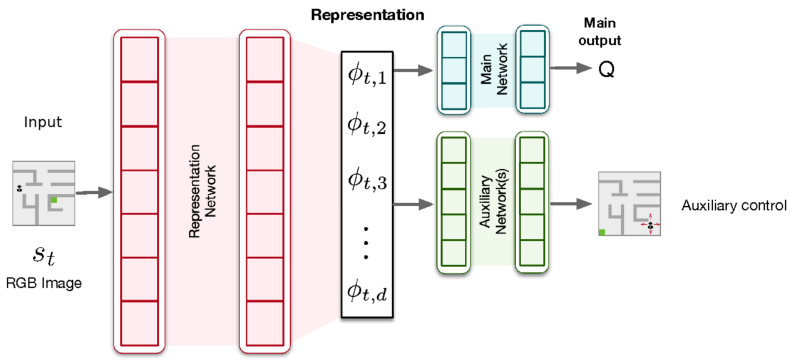

Figure 4.5: The network architecture learning representation with the auxiliary control task. The agent learns value functions based on the changed goal state or reward function in the auxiliary task. It will not affect the goal or reward in the main task.

4.2.1 Additional Goals (Simple Maze)

Our two auxiliary control tasks in Simple Maze (domain description is in Section 5.1.1) are based on the set of tasks the agent will actually face. In our experiments, we train and test the agent in environments with a different goal state, *Single Goal Control*, or with multiple goal states, called *Multiple Goals Control*. This set of goals reflects a set of possible tasks—encoded with different reward functions—with shared transition dynamics. This task encourages the representation to extract information shared by multiple value functions. Thus, if the agent could use an auxiliary loss based on one or all of these possible tasks, presumably it should learn a representation that generalizes better across tasks [24], [38], [53]. For example, we consider one auxiliary loss that uses a goal location at the center of the maze, and another set that uses five goals. The goal is to separate out the effects of using a smaller number of auxiliary control tasks, versus a more complete set. The auxiliary control tasks can be learned using Q-learning [66], which is off-policy, where the auxiliary loss is the Q-learning loss. We discuss all the auxiliary losses used for our two environments, with more details in Chapter 5.

4.2.2 Flipped Reward (Picky Eater)

Instead of adding or changing the goal state, in Picky Eater (domain description can be found in Section 5.1.2), we keep the goal state unchanged, but modify the reward function. While the main task assigns positive reward for collecting green fruits and negative reward for collecting red, the auxiliary tasks flip this. Similar to the auxiliary tasks applied to Simple Maze, this auxiliary task encourages the representation to generalize across different value functions.

Chapter 5

Experiments

The goal of this experiment is to 1) investigate the properties, listed in Table 3.1, of representations that emerge when training to good performance on a task, and 2) observe any relationships between these properties and transfer performance. We train several DQN agents with the same architectures and different auxiliary networks, as in Figure 2.1, until the agent learns a reasonable policy. We then fix this learned representation and learn the value function of a new task in the same environment. Either the goal state or the reward function is changed. Because the environment dynamics remain unchanged, the agent might be able to leverage the representation learned for transfer on the new task. Our primary goal is not to obtain effective transfer, but rather to evaluate the properties, and how they relate to transfer.

5.1 Environments and Tasks

5.1.1 Simple Maze

The first environment is a navigation environment with obstacles, called Simple Maze, which naturally enables the specification of different subgoals. The problem is episodic, with $\gamma = 0.99$. The reward is +1 when reaching the goal and 0 otherwise. The input state consists of an RGB input of a 15x15 grid (size 15x15x3), showing the agent's location and the position of the wall, but the goal state is invisible. The actions correspond to the four cardinal directions (up, down, right, left). One transition deterministically moves the agent by one pixel, or not at all if the action is into a wall. The agent starts

in a uniform random state and episodes are cut-off at 100 steps to simplify exploration, where the agent is teleported to a new random state without termination.

We define three tasks in this environment, corresponding to different goals, depicted in Figure 5.3. The agent learns first on the *original* task in which the goal state is at coordinate $[9, 9]$. In transfer learning, one task is designed to be *similar* to the original task, and one more dissimilar, in terms of the path needed to reach the goal. Their goal states are $[14, 14]$ and $[0, 14]$ respectively. For all-goals auxiliary control, tasks with goals at $[0, 0]$, $[14, 0]$, and $[7, 7]$ are used as additional auxiliary control tasks. In single-goal auxiliary control, only $[7, 7]$ is added as the goal state of the auxiliary task. Figure 5.1 shows goal states positions of all above tasks. Recall that the intuition of defining different goal state is to encourage the agent to learn different value functions as well as encoding the transition structure in the representation function, we prefer maximizing the difference between the auxiliary goals and original goal, also let the auxiliary goals have coverage over the environment, thus we choose all 4 corners and the middle of the environment as auxiliary goals.

5.1.2 Picky Eater

We design an environment that makes representation learning more difficult, and less likely to transfer to the second task. In the Picky Eater problem, depicted in Figure 5.15, the agent again observes a 15×15 RGB image, but now of a four rooms problem with two different colors of fruit: Green and Red. Four rooms are connected with openings of width 2.

Each episode begins with the 12 fruit locations randomly assigned to Red or Green, with an equal number of each. Fruits are set on 4 corners of the upper-right room and the lower-left room, as well as on the left side of each door. When the agent gets to the lower-right corner, the episode ends. Figure 5.2 shows a random observation.

Given that we have a picky eater, only one color of fruit gives a reward of $+1$; the other gives a negative reward of -1, encouraging the agent to collect a specific type of fruit while avoiding the other. In the original task, Green gives

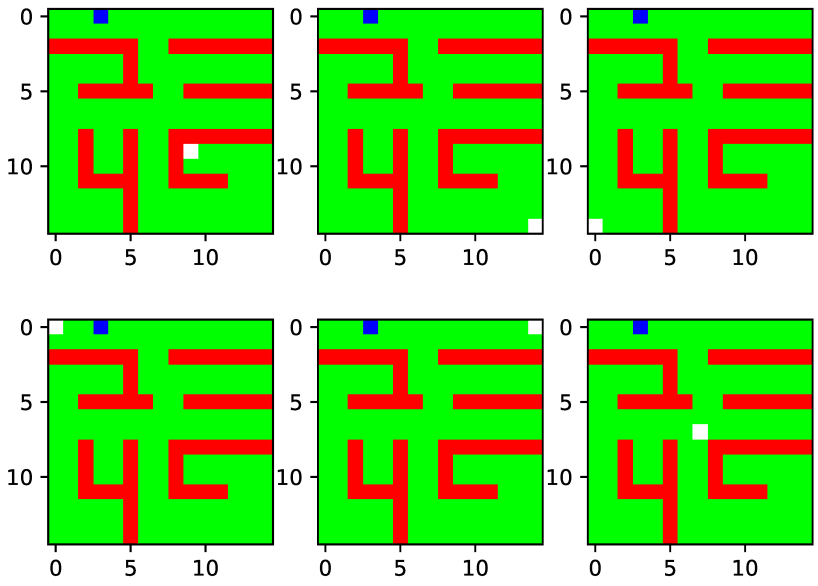

Figure 5.1: Simple maze: 6 goal states of Simple Maze. The position of the agent is shown by a blue pixel. Red blocks are walls and the background is green. The goal state is shown in white. However, it is invisible to the agent. In the plot that the agent gets, the goal state has the same color as the background. The figure on top left corner shows the goal state used in the original task. The second and the third plots on the first row are the goal states in transfer tasks. Plots on the second row show the positions of goal states added in the auxiliary control task.

+1 and Red gives −1. The agent gets a −0.001 reward each step to encourage exploration, along with the rewards given for collecting fruit. In transfer task, which only has the *dissimilar* case, the reward is flipped to encourage the agent picking up Red fruits (+1) and avoiding Green fruits (−1), while other settings including the position of exit all remain the same. This problem stresses the representation because the agent is more likely to specialize in the first task, learning specifically that Green is good and eventually simply ignoring parts of the world with Red fruit. It may not generally learn about dynamics, and that there are consumable objects that can result in a reward.

Picky eater has the same action set as Simple maze has (up, down, left, right) and it does not move the agent position when the action brings the agent into the wall. The color of a pixel which has fruit is red or green after initialization, blue during picking (that is, if the coordinate corresponds to

current agent's position), and grey once the fruit has been picked.

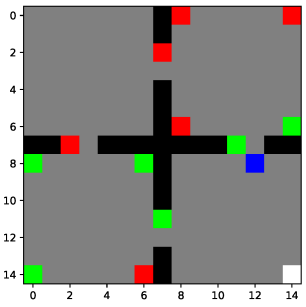

Figure 5.2: Picky Eater: the visualization of a random state. The walls are black and background is grey, the agent is blue, fruits are shown by coordinates of their respective colors (red/green), and the white pixel on the lower-right corner is the exit.

Most auxiliary tasks have same definition in both environment, which Expert Target Prediction and Auxiliary Control tasks need to be defined separately for each environment. In Table 5.2, we show more detailed definition and how auxiliary tasks are defined in each environment. In experiments, we check if any of the auxiliary tasks can encourage the representation to be useful for both tasks.

5.2 Data Collection

To evaluate the properties of the representation, we sample a batch of around 1000 transitions ($N \simeq 1000$) from the environment.

In Simple Maze, we collect data offline by doing a 100,000-step random walk, which gives 1050 episodes. We then create a dataset S_A by sampling a random transition from each episode. The dataset is used to measure Dynamics Awareness, Orthogonality, and Non-interference. Linear Probing is trained using dataset S_A. We use 40,000 steps of a random walk to collect 410 episodes, and randomly sample a transition from each episode to generate dataset S_B. We use S_B to measure Linear Probing Accuracy. This dataset is separated into the validation set and the testset equally. The validation set is used to decide when to stop training the linear probing model, while the test set is used for measuring the performance of a trained model. In Simple Maze,

a dataset S_C consisting of all possible state transition pairs is used to measure Complexity Reduction, Diversity, and Decorrelation.

In Picky Eater, we take the state value learned with the original task and collect additional data using an ϵ-greedy policy where $\epsilon = 0.1$, to make sure the dataset has more states reflecting situations when fruit being collected. The budgets for collecting data are 250,000 and 100,000 steps in this environment for S_A and S_B respectively, resulting in datasets with size 1017 and 403, with the latter being separated into validation and test sets equally. All measures are measured with S_A, except that linear probing is trained with S_A but validated and tested with S_B. Picky Eater does not have S_C.

5.3 Representation Learning Pipeline

The complete pipeline consisted of 1) training the representation network ϕ_{θ_R}, 2) control performance evaluation on the transfer tasks with a fixed representation, and 3) property evaluation. All agents used ϵ-greedy exploration, with $\epsilon = 0.1$, with standard training choices for DQN, including replay, target networks, and the Adam optimizer [27]. We saved the learned representation by saving parameters ϕ_{θ_R}. We then evaluated these fixed representations in terms of properties and for transfer performance. A new action-value was learned, from scratch, on the given task with this fixed representation, including on the original task, as a baseline comparison for transfer. All plots, therefore, including those on the original task, reported performance with this fixed representation. We included two baselines: a Random representation and Baseline-scratch. Random used the randomly initialized NN, without any learning. Baseline-scratch was a DQN agent trained from scratch on that task, including learning the representation (without auxiliary losses).

We utilized the same neural network architecture across representations learned with the various loss functions, for both environments. The obtained representations were in $\mathbb{R}^{32}$, with details of the architectures as given un Table 5.1. During training, all inputs were normalized to be in the range -1 to 1. We used the ADAM optimizer to update weights. Nonlinear layers used ReLU

activation function. For Expert Targets auxiliary loss – we used cross-entropy as the loss function for predicting the color of a pixel, while other tasks used mean-squared error.

Table 5.1: Representation learning network structures.

Environment	**layer**	**input**	**output**	**kernel**	**stride**	**pad**
Simple Maze	1	3	16	4	2	2
	2	16	8	4	2	2
Picky Eater	1	3	32	4	1	1
	2	32	16	4	2	2

In Simple Maze, the agent was trained for a fixed budget of 300,000 steps. The agent with Input-decoder was trained longer, with an additional 200,000 steps, for it to obtain reasonable performance. In each step, 32 samples were randomly chosen from a buffer of memory size 10,000. The target net was updated every 64 steps. We chose the best learning rate from the list $[0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001]$ based on the averaged result of 5 runs. Then we performed 60 runs with the picked learning rate to report the average number and standard error (std). We set a timeout threshold for the agent, thus the episode ends either when the agent gets to the goal state or when the number of steps of an episode reaches 100.

In Picky Eater, most experimental details are identical to Maze World, except that we increased the learning time to 1,000,000 steps, applied a slower target network synchronization (every 1024 steps), a larger buffer containing 100,000 samples, and a larger timeout threshold (500). We used the same strategy for picking the learning rate as in Simple Maze, then performed 30 runs with the best learning rate. Besides, for this problem, the goal state does not change. There was only one goal in the Auxiliary Control task (Pick Red with $\gamma = 0.99$), as opposed to the five in Maze World.

The value network of each environment consisted of 2 fully connected layers. We used 64 nodes on each layer in Simple Maze and 128 nodes in Picky Eater. Auxiliary networks had different structures because of the capacity requested by different tasks. We indicate what these auxiliary tasks do in Table 5.2 and their architecture in Table 5.3. The main-task value network and the auxiliary

network were updated together with the representation network. The learning rate chosen after the parameter sweep for each representation is shown in Table 5.4. Note that the γ used in Single-goal control, All-goals control, and Successor-feature prediction was 0.99.

Table 5.2: Auxiliary tasks explanation and environments they are applied to. Expert target prediction and auxiliary control tasks are defined separately for each environment. Single-goal and all-goals control are only defined for Simple Maze, while Expert-color, Expert-count, and Pick-red tasks are limited to the Picky Eater environment.

Representation	**Explanation**	**Environment**
Input-decoder	Predict the input image	Simple Maze & Picky Eater
Expert-xy Prediction	Predict the current position of the agent	Simple Maze & Picky Eater
Expert-(xy+color) prediction	Keep the prediction of the current position since it is an important information for the agent to know, also add the prediction of fruits' color	Picky Eater
Expert-(xy+count) prediction	Keep the prediction of the current position since it is an important information for the agent to know, also add the prediction of the number that green and red fruits left	Picky Eater
Single-goal control	Learn the value function of one auxiliary goal state on the auxiliary head	Simple Maze
All-goals control	Learn value functions of 5 auxiliary goal states on the auxiliary heads	Simple Maze
Pick-red control	Learn the value function given the flipped reward function on the auxiliary head	Picky Eater
Next-agent-state prediction	Predict the next agent state	Simple Maze & Picky Eater
Successor-feature prediction	Predict the successor feature of the current agent state	Simple Maze & Picky Eater

Table 5.3: Auxiliary tasks network structures.

Representation	**Auxiliary network structure (Simple Maze)**	**Auxiliary network structure (Picky Eater)**
Input-decoder	1 fully connected layer with 200 nodes 2 transposed convolutional layers layer 1: input channel: 8, output channel: 16, kernel: 4, stride: 2, pad: 2 layer 2: output channel: 3, kernel: 4, stride: 2, pad: 2	1 fully connected layer with 1024 nodes 2 transposed convolutional layers layer 1: input channel: 16, output channel: 32, kernel: 4, stride: 2, pad: 0 layer 2: output channel: 3, kernel: 4, stride: 1, pad: 0
Expert-xy prediction	2 fully connected layers with 64 nodes on each	2 fully connected layers with 128 nodes on each
Expert-(xy+color) prediction		2 fully connected layers with 128 nodes on each
Expert-(xy+count) prediction		2 fully connected layers with 128 nodes on each
Single-goal control	2 fully connected layers with 64 nodes on each	
All-goals control	2 fully connected layers with 64 nodes on each	
Pick-red control		2 fully connected layers with 128 nodes on each
Next-agent-state prediction	2 fully connected layers with 64 nodes on each	2 fully connected layers with 128 nodes on each
Successor-feature prediction	3 fully connected layers with 512 nodes on each	3 fully connected layers with 512 nodes on each

Table 5.4: Representation learning rate. The learning rate is swept in a fixed list then chosen based on the best performance over 5 runs. The best learning rate in Simple Maze is either 0.0001 or 0.0003, while in Picky Eater, 0.00003 always performs the best.

Representation	**Simple Maze**	**Picky Eater**
No auxiliary	0.0001	0.00003
Input-decoder	0.0003	0.00003
Expert-xy prediction	0.0001	0.00003
Expert-(xy+color) prediction		0.00003
Expert-(xy+count) prediction		0.00003
Single-goal control	0.0003	
All-goals control	0.0001	
Pick-red control		0.00003
Next-agent-state prediction	0.0003	0.00003
Successor-feature prediction	0.0003	0.00003

5.4 Details about Measured Properties

Complexity Reduction: We used S_C in Simple Maze and S_A in Picky Eater as the test set when measuring this property. For every 20,000 and 50,000 steps in Simple Maze and Picky Eater representation learning steps respectively, we checked the action values for all states in the dataset and calculate the measure according to the formula in Table 3.1.

Linear Probing: We used S_A to train the linear network with ADAM optimizer, cross-entropy loss for color prediction, and mean-squared error (MSE) loss for other tasks. The learning rate is swept and chosen using the same pipeline as representation learning, except the list for sweeping is $[0.1, 0.01, 0.001, 0.0001, 0.00001]$. At each step, we randomly sampled 32 samples from the data set and update the linear network. Every 100 steps, we tested the linear network on the validation set extracted S_B. The training was cut-off if the validation loss does not decrease for 10 consecutive tests. The error measure reported is percentage error $1 - \sum_i \frac{|y_i - \hat{y}_i|}{1+|y_i|}$ (when loss is MSE) and accuracy $\frac{n_{correct\_prediction}}{n_{total\_sample}}$ (when loss is cross-entropy), measured using the test set which had around 200 samples. The learning rate used is shown in Table 5.5.

In Simple Maze, we evaluated the learned representations in terms of their

ability to linearly predict (x, y) position of the agent (the measure is called *linear probing - xy*) (Section 3.1.3 and Table 5.6). In Picky Eater, we additionally evaluated the accuracy of linear predictions of the fruit color on position $(2, 7)$ (the measure is called *linear probing - color*), as well as linear predictions of the numbers of red fruits and green fruits (the measure is called *linear probing - count*). For *linear probing - xy* and *linear probing - count* evaluations, we report percentage error. For *linear probing - color* evaluation, we report accuracy.

Dynamics Awareness: We took all one-step transitions (s_t and s_{t+1}) from S_A as similar state pairs, which should give small distance as they are temporally close. For every s_t in the dataset, we randomly sample another s_j to formulate a different state pair, which should give high distance. The data reported are calculated using the formula given in Table 3.1.

Diversity (1−Specialization): We took all possible pairs from dataset S_A and the trained value network, then calculate the diversity according to Table 3.1.

Orthogonality: We took all possible pairs from dataset S_A, then calculate the orthogonality according to Table 3.1.

Decorrelation: We took all possible pairs of features from the representation, then use states in S_A to calculate the decorrelation according to Table 3.1.

Non-Interference: We took all possible pairs from dataset S_A, then calculate the non-interference according to Table 3.1. We use TD-error as δ in this measure.

5.5 Experimental Results

In this section, we provide results on the two environments. We first show learning curves on the original task, then the transfer task and finally summarize the properties of learned representations.

Table 5.5: Linear Probing learning rate. The learning rate is swept in a fixed list then chosen based on the best performance over 5 runs. The best learning rate varies from 0.1 to 0.00001, depending on the representation and the target that it predicts, though Input-decoder representation, Pick-red control representation, and Next-agent-state prediction representation have the same best learning rate (0.001) for all tagets.

Representation	**XY prediction (Simple Maze)**	**XY prediction (Picky Eater)**	**Color prediction (Picky Eater)**	**# Red/Green Fruits prediction (Picky Eater)**
No auxiliary	0.1	0.001	0.001	0.001
Input-decoder	0.001	0.001	0.001	0.001
Expert-xy prediction	0.001	0.1	0.1	0.1
Expert-(xy+color) prediction		0.0001	0.0001	0.0001
Expert-(xy+count) prediction		0.001	0.0001	0.0001
Single-goal control	0.1			
All-goals control	0.01			
Pick-red control		0.001	0.001	0.001
Next-agent-state prediction	0.001	0.001	0.001	0.001
Successor-feature prediction	0.00001	0.1	0.1	0.001
Random representation	0.1	0.01	0.1	0.1

5.5.1 Simple Mazes

Performance in the Original Task

Here we show the learning curve when learning the representation, with parameters described in Section 5.3. Except for Random-representation which takes a randomly initialized neural network as the representation network, all other agents converge to a reasonable policy. This is because the average return equals or is close to 1, so the learning curves of them do not show a large difference. At the end of this experiment, the representations were stored to later be used for a new initialization of DQN to evaluate the learned representations.

Transfer Performance

Figure 5.4 shows the transfer performance. Several auxiliary losses result in a representation that is useful for efficient learning in the original task and the similar task, but only compared to learning everything from scratch. They

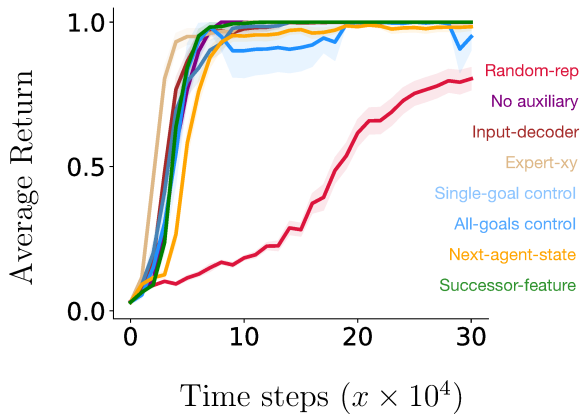

Figure 5.3: Representation learning in Simple Maze: Learning curves for each DQN agent as it trained in the *original task* Simple Maze. Plotted is the average return, averaged over 60 runs of the experiment. The shaded region in the plot indicates ±2 std.

are not better than the representation learned by No auxiliary (DQN without auxiliary tasks). This changes once moving to the different task. There, No auxiliary only provides slight improvements on Baseline-scratch. Successor-features, Single-goal and All-goal Control and Expert-xy all improve on *No auxiliary* in the dissimilar task. As expected, Input-decoder was never competitive in any of the transfer tasks, performing similarly to a Random representation.

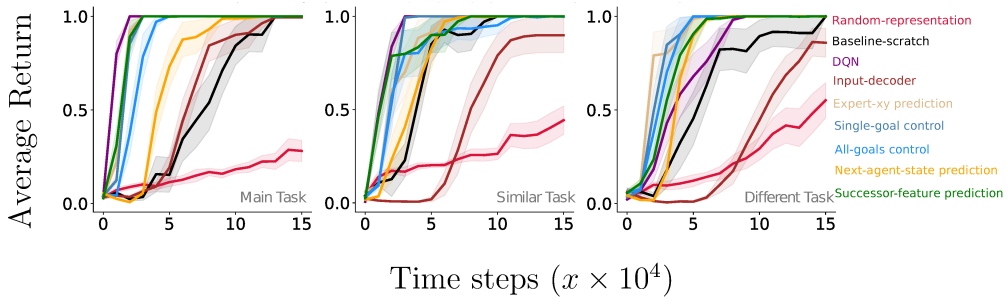

Figure 5.4: Simple Maze transfer performance. Learning curves summarize the performance of fixed, pre-learned representations on three tasks: (left) *original* task that the representations were trained on, (middle) the *similar* task, (right) the *dissimilar* task. In all three cases the representations trained in experiment shown in Figure 5.3 were used to initialize the representation layers fresh DQN agent. The representations were not adapted during the experiment, only the value network weights were adjusted, representing a pure representation transfer setting. Most representations exhibit good transfer compared with DQN (with full representation learning from scratch). Notice that in the hardest transfer task the representations learned from auxiliary losses outperform the representation without auxiliary losses (pre-trained DQN)—this is noteworthy because the auxiliary losses were disabled during the transfer experiments.

Property Measures

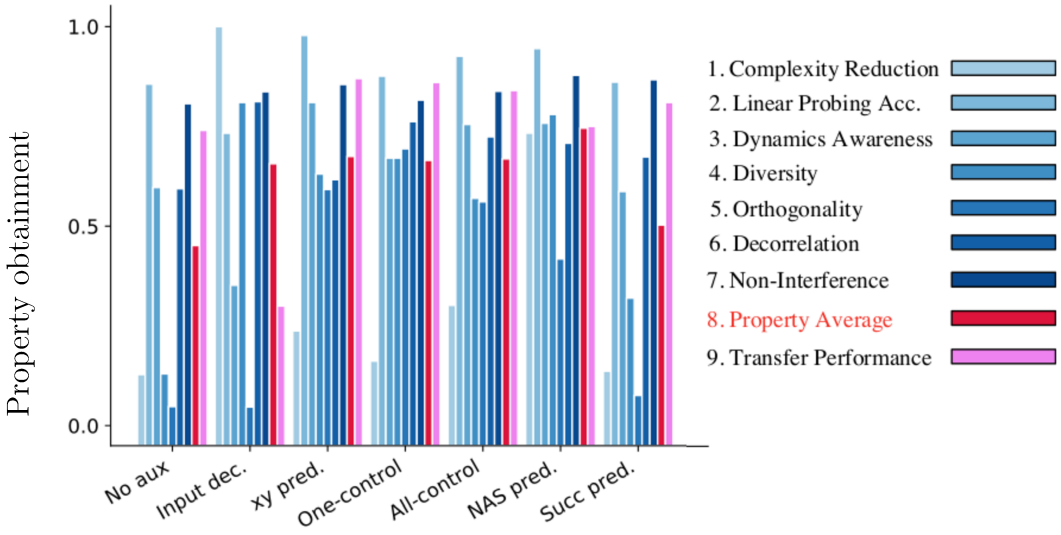

Figure 5.5: Property obtainment for representations induced by different auxiliary losses in Simple Maze. The red bar indicates the average normalized property per architectures. These visualizations are meant to give a birds-eye view of all the properties, to aid in comparison to the transfer performance. A table of all properties is in Figure 5.6.

Simple Maze	Transfer: Same Goal		Transfer: Similar Goal		Transfer: Different Goal		Task independent properties				
Architecture	Complexity Reduction	Specialization	Complexity Reduction	Specialization	Complexity Reduction	Specialization	Decorrelation	Dynamics Awareness	Linear Probing Accuracy (XY)	Non-Interference	Orthogonality
Random representation	0.12 (0.083)	0.06 (0.025)	0.05 (0.089)	0.03 (0.033)	0.10 (0.118)	0.08 (0.020)	0.114 (0.008)	0.250 (0.024)	0.716 (0.015)	0.836 (0.027)	0.011 (0.005)
Baseline-scratch	0.92 (0.005)	0.86 (0.011)	0.92 (0.002)	0.85 (0.014)	0.92 (0.006)	0.81 (0.030)					
No Auxiliary	0.94 (0.002)	0.87 (0.008)	0.93 (0.03)	0.79 (0.010)	0.84 (0.008)	0.18 (0.009)	0.406 (0.063)	0.597 (0.030)	0.856 (0.015)	0.807 (0.038)	0.048 (0.019)
Input-decoder	0.99 (0.000)	0.19 (0.028)	0.98 (0.002)	0.19 (0.039)	0.97 (0.008)	0.04 (0.026)	0.188 (0.034)	0.352 (0.041)	0.733 (0.017)	0.837 (0.020)	0.047 (0.014)
Expert-xy prediction	0.96 (0.002)	0.37 (0.009)	0.96 (0.002)	0.43 (0.009)	0.96 (0.001)	0.47 (0.011)	0.384 (0.027)	0.810 (0.015)	0.978 (0.003)	0.855 (0.016)	0.592 (0.019)
Single-goal control	0.95 (0.002)	0.33 (0.012)	0.95 (0.002)	0.38 (0.017)	0.91 (0.004)	0.19 (0.019)	0.238 (0.023)	0.670 (0.023)	0.876 (0.012)	0.816 (0.022)	0.694 (0.023)
All-goals control	0.95 (0.002)	0.43 (0.018)	0.97 (0.003)	0.44 (0.039)	0.96 (0.002)	0.54 (0.027)	0.276 (0.041)	0.755 (0.016)	0.926 (0.022)	0.838 (0.017)	0.561 (0.195)
Next agent-state prediction	0.98 (0.004)	0.22 (0.046)	0.98 (0.002)	0.38 (0.060)	0.99 (0.001)	0.24 (0.020)	0.292 (0.043)	0.758 (0.025)	0.945 (0.017)	0.878 (0.033)	0.418 (0.158)
Successor-feature prediction	0.94 (0.002)	0.68 (0.014)	0.94 (0.002)	0.48 (0.022)	0.87 (0.007)	0.16 (0.009)	0.327 (0.033)	0.587 (0.028)	0.861 (0.016)	0.867 (0.018)	0.076 (0.016)

Figure 5.6: Simple Maze: Raw properties data corresponding to the measures defined Table 3.1. Baseline-scratch (black color) is a baseline added in transfer tasks only, because it is as same as No Auxiliary case in the original task. Thus there is no data for Baseline-scratch when measuring the task independent properties.

Next, we evaluate the properties of these representations. A summary of all properties measures is given in Figure 5.5. Below we give more detailed results. The violin plots show the mean value (dash in the middle), maxima and minima (dots on each bubble), and the data distribution (shadow). The description and analysis are below each plot.

When looking at the summary in Figure 5.5, there are two relatively clear groupings in terms of properties. The first group has both a relatively good transfer performance and a relatively high averaged property score—the Control tasks (One-control and All-control), Next-agent-state (NAS pred) and Expert-xy (xy pred) all resulted in generally higher level of the desirable properties, in the meanwhile, they all perform the best on the *dissimilar task* transfer. When focusing on the score of each property, we find that representations from this group generally exhibit high orthogonality (see also Figure 5.12), high linear probing accuracy (see also Figure 5.11) and high dynamics awareness (see also Figure 5.10). The notable differences are in complexity reduction on the *dissimilar task*, where Next-agent-state scores high and Single-goal control scores low (Figure 5.7-c).

The second group consists of No auxiliary (No aux), Input-decoder (Input dec), and Successor-features (Succ pred). They have relatively lower transfer score. The profiles for No auxiliary and Successor-features are very similar, and they perform quite similarly on the control tasks. The main difference is that Successor-features has slightly higher decorrelation and slightly higher control performance. Input-decoder has different properties than No auxiliary, with notably higher complexity reduction and diversity (see also Figure 5.8) and lower linear probing accuracy and lower dynamics awareness. Input-decoder has low orthogonality but high decorrelation, suggesting orthogonality might be more predictive of transfer performance. Finally, non-interference is also high for all methods; there are some differences, though it does not seem to have much correlation with performance, even when only considering the better agents.

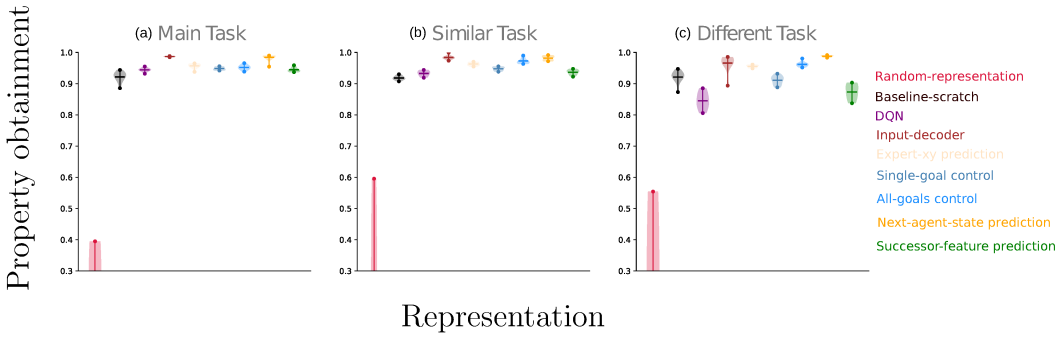

Figure 5.7: **Complexity Reduction in Simple Maze.** The measure of complexity reduction is related to the learned value function, which is different in different transfer tasks. The plot shows the measure in three transfer tasks—the one as same as the original task, the one similar to the original task, and the one different from the original task. Auxiliary tasks generally improve representations in terms of complexity reduction. The improvement is stark for *Next-agent-state* and marginal for *Expert-xy prediction*. Notice the decrease in *No auxiliary's* and *Successor-Feature's* complexity reduction as the representation is used to solve *dissimilar task*.

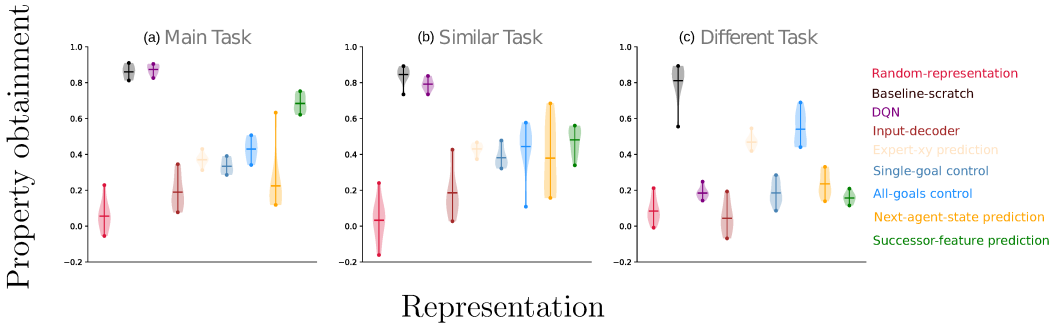

Figure 5.8: **Specialization in Simple Maze.** The measure of specialization is related to the learned value function, which is different in different transfer tasks. The plot shows the measure in three transfer tasks—the one as same as the original task, the one similar to the original task, and the one different from the original task. *No Auxiliary* resulted in representations that are specialized to the original task (less diverse) – specialization score drops significantly as the representation is used to solve *dissimilar task*. In contrast, auxiliary tasks resulted in less specialized (more diverse) representations.

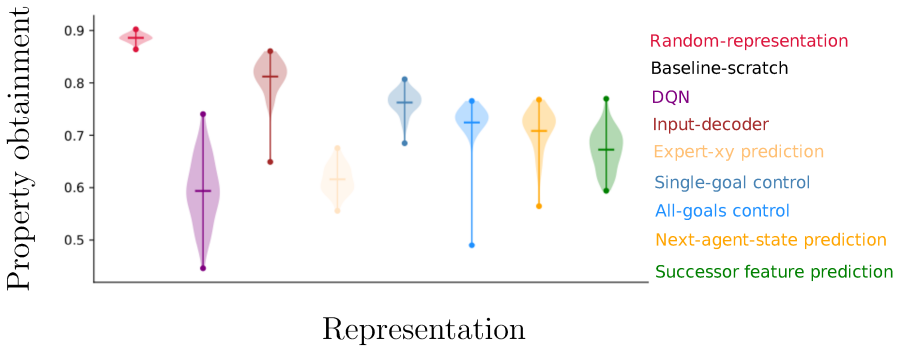

Figure 5.9: **Decorrelation in Simple Maze.** Auxiliary tasks improve representations in terms of decorrelation.

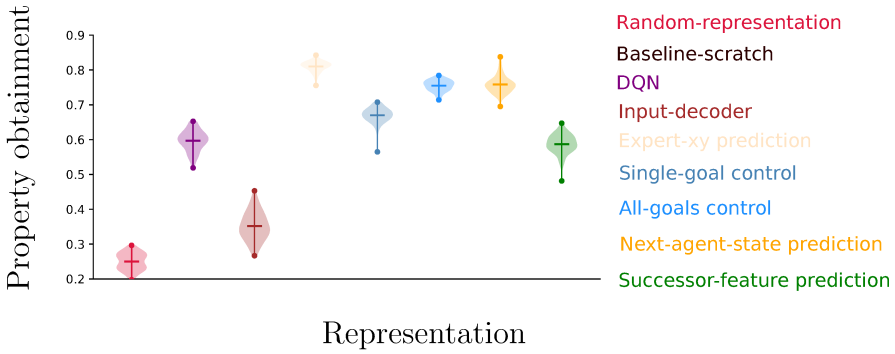

Figure 5.10: **Dynamics Awareness in Simple Maze.** *Next-agent-state, Expert-xy prediction,* and auxiliary control tasks improved Dynamics Awareness relative to *No Auxiliary*; *Input-decoder* resulted in representations with poor Dynamics Awareness.

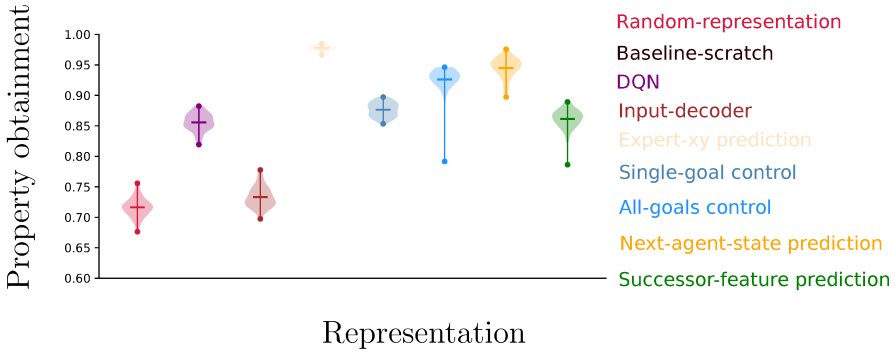

Figure 5.11: **Linear Probing in Simple Maze.** Except for *Input-Decoder* and *Successor-features*, auxiliary tasks improve linear probing scores of the learned representations. *Input-Decoder* reduces the linear probing score – while the representation can be used to successfully decode the input state (using a decoder network), it cannot be used to linearly predict the position of the agent with high accuracy.

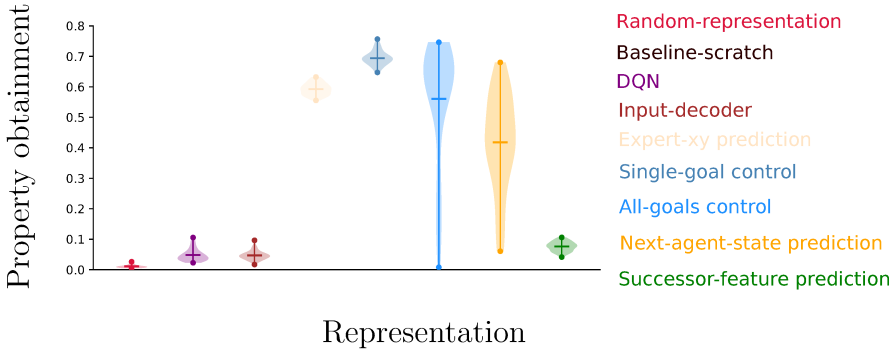

Figure 5.12: **Orthogonality in Simple Maze.** With the exception of *Input-Decoder*, auxiliary tasks improve orthogonality of the learned representations. The improvement is stark for *Next-agent-state*, *Expert-xy prediction*, and auxiliary control tasks. It is marginal for *Successor-features*.

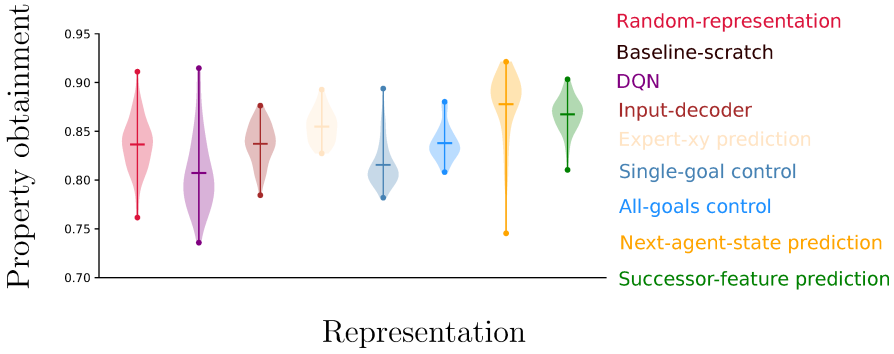

Figure 5.13: **Non-interference in Simple Maze.** Auxiliary tasks improve non-interference only marginally relative to *No auxiliary*.

5.5.2 Picky Eater

Performance in the Original Task

Figure 5.14 shows learning curves when we train the representation in Picky Eater. Compared to the Simple Maze, Picky Eater is a harder environment thus the agent takes longer to learn a good policy. There are always 6 fruits that give positive reward. By collecting all of them and avoiding the other color, the agent gets +6, which is an upper threshold of the return. Because of the random start, the random color initialization, and the negative reward that the agent gets at each timestep, the maximum return of each episode changes slightly, but they are generally between 5 and 6. The 1×10^6-timestep budget is enough for most agents (No auxiliary, Next-agent-state prediction, Successor-feature prediction, Expert-xy prediction and Input-decoder) to learn a reasonable policy to gain a high return. At the end of this experiment, the representations are stored to be used later for a new initialization of DQN to evaluate the learned representations in Pick Green and Pick Red tasks.

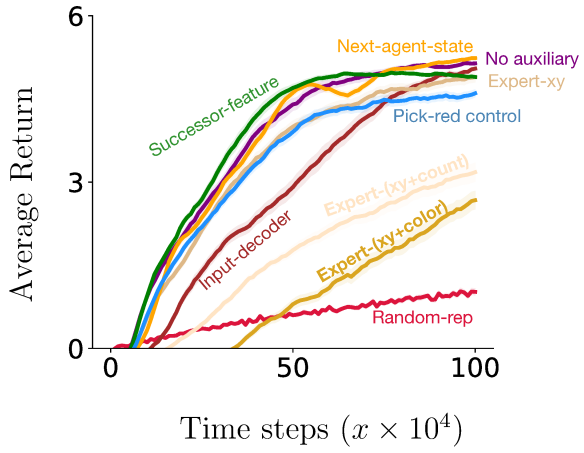

Figure 5.14: Representation learning in Picky Eater: Learning curves for each DQN agent as it trained in the *Pick Green* task. Plotted is the average return, averaged over 30 runs of the experiment. The interval shows 2 times of the standard error.

Transfer Performance

Except for testing the transfer performance of fixed representation, we also test performance when fine-tuning the representation on the second task. This complements the question of if the learned representation transfers to the second task, to also ask if the representation allows for faster future learning.

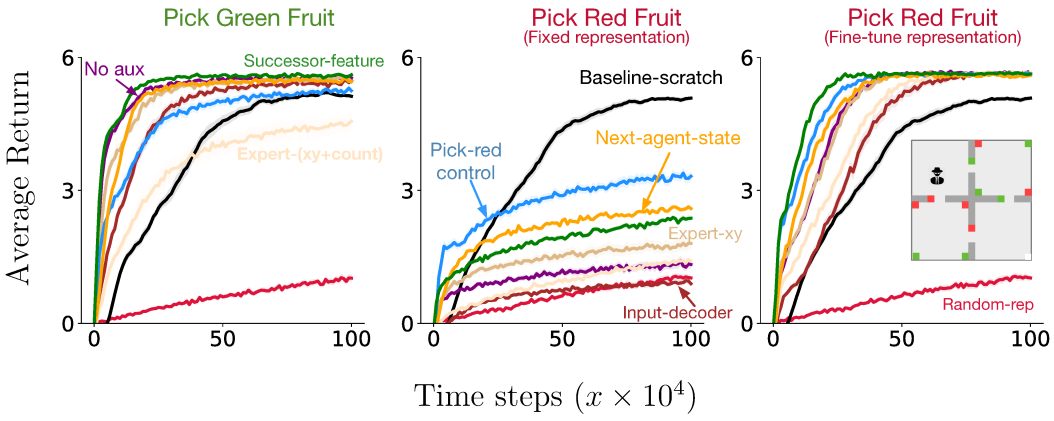

Figure 5.15: Learning curves for fixed, pre-learned representations on the original pick green task the representation was trained on, and pick red task. The rightmost plot shows the impact of fine-tuning the representation learned on pick red. Results are averaged over 30 runs, with standard error bars.

The control performance of 3 transfer tasks is shown in Figure 5.15. As in Simple Maze, No auxiliary performs well on the original task, and most of the representations learn a reasonable policy, though not better than No auxiliary. In the Pick Red Fruit task, as expected, the representations transfer more poorly because of the change on the reward function: Baseline-scratch surpasses all fixed representations eventually. The learned representations, however, do enable faster initial learning and still manage to obtain a return of 3, meaning they do collect the right fruit. Auxiliary control and Successor-features again achieve the highest performance. In contrast to Maze World, Next-agent-state is considerably better than Expert-xy; this makes sense in this environment, where xy-coordinates is not as informative and the next state is useful to know across both tasks.

Fine-tuning the representation considerably improves performance, while still maintaining the early learning advantages of the learned representations.

Interestingly, the relative ordering of representations is almost the same under fine-tuning, except that Successor-features vaults to the top.

Property Measures

Before going to the score of each property, we put the summary of the properties for these representations in Figure 5.16. Auxiliary control (control), Next-agent-state (NAS pred), and Successor-feature prediction (Succ pred) are highly performant methods regarding to the transfer learning. Different from Simple Maze, this group does not show a clear better averaged property score. When focusing on each property, we notice that orthogonality reflects the transfer performance ordering in this group; decorrelation is not as clear and is typically high for most agents. The complexity reduction measures are reasonably predictive of positive representation transfer (Figure 5.18). The cases with good complexity reduction and poor performance are due to small values due to lack of learning (Figure 5.15-center and 5.18). This suggests a natural improvement of this measure to distinguish these two cases. Dynamics awareness is uninformative in this environment—potentially it is comprised of open rooms—and most methods scored highly on it.

The conclusions are different from Simple Maze, particularly because failures from certain agents skew patterns. For example, the higher complexity reduction of Expert-targets and Input-decoder seems to be due to their poor transfer performance (Figure 5.15-center): the learned values are small and result in a smaller Lipschitz constant. For the other methods, complexity reduction did correlate with performance. This suggests a natural improvement of this measure to distinguish these two cases.

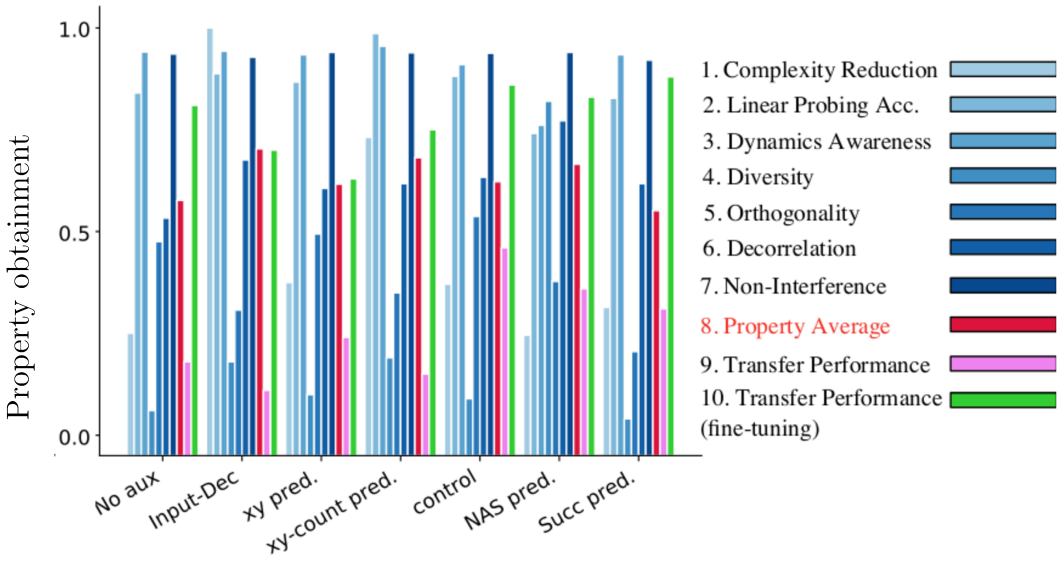

Figure 5.16: Property obtainment for representations induced by different auxiliary losses in Picky Eater. The red bar indicates the average normalized property per architectures. These visualizations are meant to give a birds-eye view of all the properties, to aid in comparison to the transfer performance. A table of all properties is in Figure 5.17

Picky Eater	Transfer: Same Goal (Pick Green)		Transfer: Different Goal (Pick Red)		Task independent properties						
Architecture	Complexity Reduction	Specialization	Complexity Reduction	Specialization	Decorrelation	Dynamics Awareness	Linear Probing Accuracy (XY)	Linear Probing Accuracy (color of block)	Linear Probing Accuracy count colored blocks	Non-Interference	Orthogonality
Random representation	0.03 (0.08)	0.35 (0.026)	0.08 (0.07)	0.33 (0.021)	0.184 (0.004)	0.710 (0.029)	0.705 (0.013)	0.947 (0.027)	0.674 (0.042)	0.914 (0.099)	0.115 (0.048)
Baseline-scratch	0.78 (0.009)	0.94 (0.005)	0.79 (0.011)	0.93 (0.005)							
No auxiliary	0.71 (0.021)	0.94 (0.004)	0.30 (0.042)	0.44 (0.062)	0.467 (0.009)	0.941 (0.006)	0.872 (0.016)	0.875 (0.058)	0.774 (0.031)	0.936 (0.007)	0.475 (0.031)
Input-decoder	0.93 (0.001)	0.82 (0.010)	0.84 (0.014)	0.60 (0.040)	0.324 (0.004)	0.943 (0.009)	0.871 (0.018)	0.990 (0.003)	0.802 (0.020)	0.928 (0.009)	0.307 (0.028)
Expert-xy prediction	0.81 (0.009)	0.90 (0.010)	0.55 (0.030)	0.62 (0.033)	0.394 (0.009)	0.934 (0.007)	0.956 (0.018)	0.882 (0.077)	0.763 (0.041)	0.940 (0.013)	0.494 (0.041)
Expert-(xy+color) prediction	0.94 (0.002)	0.69 (0.008)	0.90 (0.014)	0.61 (0.016)	0.249 (0.003)	0.952 (0.003)	0.993 (0.001)	0.992 (0.003)	0.797 (0.018)	0.942 (0.010)	0.488 (0.035)
Expert-(xy+count) prediction	0.88 (0.003)	0.81 (0.015)	0.85 (0.007)	0.73 (0.020)	0.382 (0.004)	0.955 (0.003)	0.992 (0.002)	0.971 (0.020)	0.996 (0.000)	0.939 (0.015)	0.350 (0.053)
Pick-red control	0.79 (0.004)	0.91 (0.007)	0.64 (0.015)	0.56 (0.053)	0.366 (0.007)	0.910 (0.009)	0.893 (0.008)	0.952 (0.039)	0.799 (0.022)	0.938 (0.012)	0.537 (0.037)
Next-agent-state prediction	0.71 (0.007)	0.18 (0.006)	0.34 (0.045)	0.17 (0.009)	0.228 (0.002)	0.761 (0.008)	0.960 (0.006)	0.554 (0.043)	0.709 (0.013)	0.940 (0.012)	0.378 (0.024)
Successor-feature prediction	0.77 (0.013)	0.96 (0.003)	0.45 (0.030)	0.35 (0.059)	0.382 (0.004)	0.934 (0.005)	0.864 (0.030)	0.871 (0.076)	0.747 (0.067)	0.921 (0.015)	0.205 (0.020)

Figure 5.17: Picky Eater: Raw properties data corresponding to the measures defined Table 3.1. Baseline-scratch (black color) is a baseline added in transfer tasks only, because it is as same as No Auxiliary case in the original task. Thus there is no data for Baseline-scratch when measuring the task independent properties.

More details are given in violin plots.

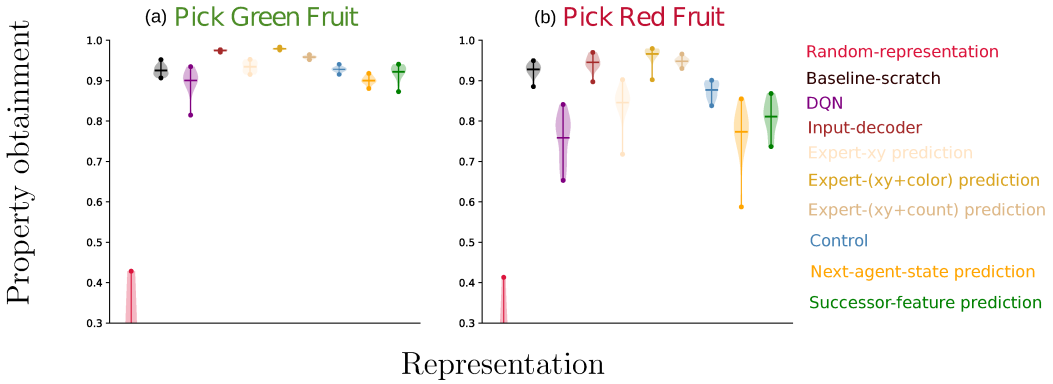

Figure 5.18: **Complexity Reduction in Picky Eater.** The measure of complexity reduction is related to the learned value function, which is different in different transfer tasks. The plot shows the measure in two transfer tasks. Auxiliary tasks generally improved representations in terms of complexity reduction. On *pick red* task, while complexity reduction of all methods decreased, representations with auxiliary tasks still scored higher than *No Auxiliary*.

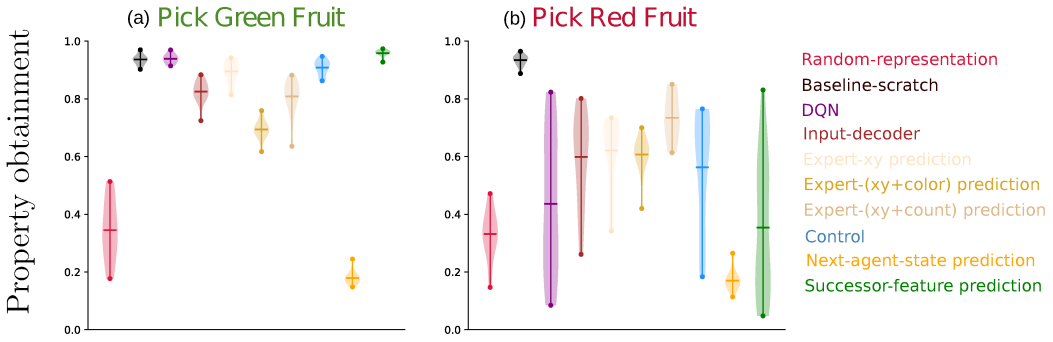

Figure 5.19: **Specialization Picky Eater.** The measure of complexity reduction is related to the learned value function, which is different in different transfer tasks. The plot shows the measure in two transfer tasks. *No Auxiliary* resulted in representations that are specialized to the original task (less diverse). Specialization score drops significantly as the representation is used to solve the dissimilar task (*Pick red fruit*). In contrast, the auxiliary task results in less specialized (more diverse) representation.

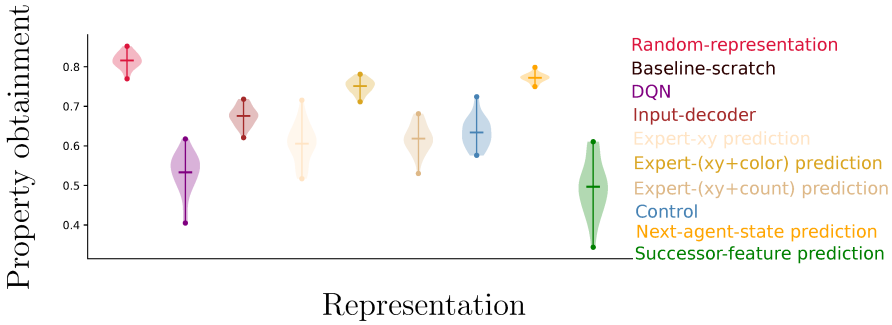

Figure 5.20: **Decorrelation in Picky Eater.** *Input-decoder*, *Expert targets*, *Pick Red Control*, and *Next-agent-state* improved decorrelation over *No auxiliary*, whereas *successor-feature prediction* performed slightly worse than *No auxiliary*.

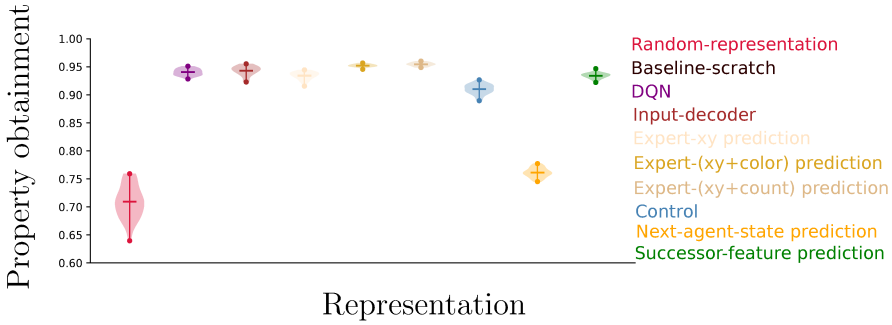

Figure 5.21: **Dynamics Awareness in Picky Eater.** *Next-agent-state* is particularly less dynamics aware than *No Auxiliary*. The remaining methods do not improve representations in terms of Dynamics Awareness when compared with *No auxiliary*.

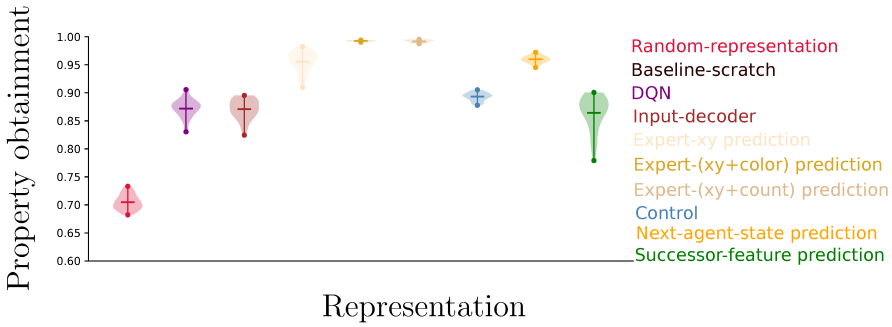

Figure 5.22: **Linear Probing Accuracy (xy) in Picky Eater.** Except for *Input-Decoder* and *Successor-features*, auxiliary tasks improve the linear probing (xy) score over *No auxiliary*.

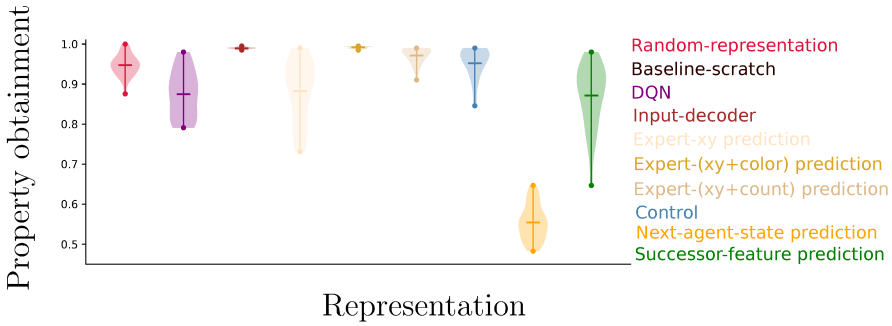

Figure 5.23: **Linear Probing Accuracy (Color) in Picky Eater.** *Input-Decoder*, *Expert-(xy+color)*, *Expert-(xy+count)*, and *Pick-red control* improve the linear probing accuracy for predicting the color of fruits. Interestingly, while *Next-agent-state* improved linear probing accuracy for xy prediction, it performed particularly worse than other methods for fruit color prediction. Note that this linear probing accuracy property checks the prediction accuracy of the color of fruit only, while Expert-(xy+color) and Expert-(xy+count) auxiliary task keep the prediction on agent position, since the position is considered as an important information for the agent to know.

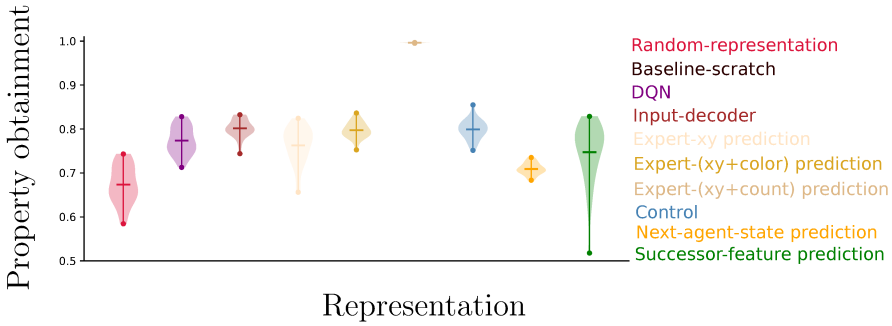

Figure 5.24: **Linear Probing Accuracy (Count) in Picky Eater.** With the exception of *Expert-(xy+count)*, auxiliary tasks do not improve representations in terms of linear probing of fruit count. Note that this linear probing accuracy property checks the predict accuracy of the number of fruits left only, while Expert-(xy+color) and Expert-(xy+count) auxiliary task keep the prediction on agent position, since the position is considered as an important information for the agent to know.

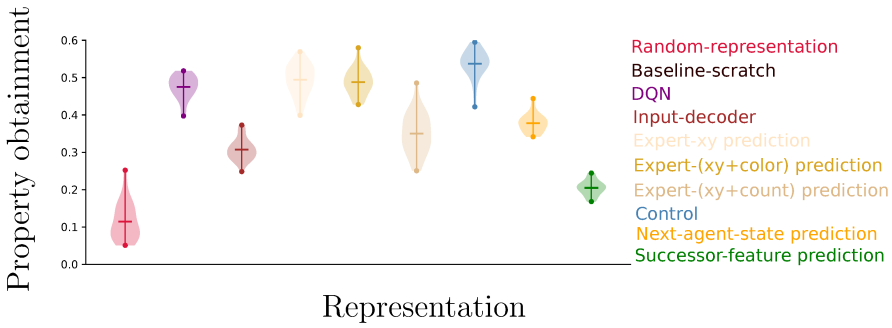

Figure 5.25: **Orthogonality in Picky Eater.** While *Pick-red* control improves orthogonality, *successor-feature prediction* most significantly hurts orthogonality.

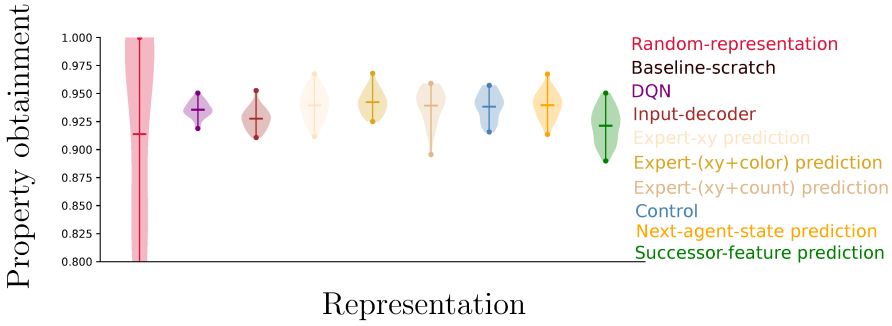

Figure 5.26: **Non-Interference in Picky Eater.** Auxiliary tasks do not appear to affect representations in terms of Non-interference in this problem setting.

5.6 Summary Analysis Across Environments

Given the above results in both environments, we try to analyze whether representations show consistent performance in different environments. Regarding a given property measure, some representations have consistent performance in both environments while others do not. There are also some measures that do not have any strong relationship with the performance in the transfer. Below we list points we see in our experiments.

1. One common result in both Simple Maze and Picky Eater is that *some* auxiliary tasks do help with transfer learning, but not all of them (Figure 5.4 and 5.15). **Adding auxiliary tasks to predict the successor-feature or the next-agent-state, or adding auxiliary control, consistently helps with transfer learning for a relatively different task comparing to the original task.** On the contrary, adding a decoder for reconstructing the observation image was consistently bad at transfer. This auxiliary task is also worse than predicting expert knowledge such as coordinate of the agent, the color of the fruit, and the number of fruits left. Adding the decoder forces the agent to include every detail in the observation, even part of them are useless. Thus, one guess is that such a constraint requires a larger capacity in the representation than necessary and hurts the representation's ability to extract useful information.

2. When measuring orthogonality, we see that **representations with a relatively high orthogonality score always show a good performance in transfer without fine-tuning**, while a representation with a good transfer performance does not imply a high orthogonality score (Figure 5.5, 5.12, 5.16, and 5.25). The guess is that except for orthogonality, there are also other properties that help with improving transfer performance, so that representations can still gain advantage in transfer learning even without high orthogonality. However, this result may still suggest a potential possibility of predicting transfer performance with orthogonality.

3. Auxiliary tasks generally improve, or at least not hurt, the representation in

terms of the complexity reduction and decorrelation, regardless of whether the transfer performance is improved or not (Figure 5.7, 5.9, 5.18, and 5.20). But **there does not appear to be a clean relationship between the measures of these two properties and the transfer performance**, especially that *Input-decoder* gains a high decorrelation score while performing bad in transfer at the same time.

4. **Without an auxiliary task, the representation highly specializes to the original task.** The score drops when the task changes. When there is a large difference between the original and transfer tasks, the scores for all representations drops; the score for *No auxiliary* representation drops more than the others (Figure 5.8 and 5.19). With respect to the performance of the same representation in different transfer tasks, when the representation learns slower (has smaller area under the curve) in one task comparing to the other, the specialization score is also lower than the score in the other. This suggests that **the specialization (diversity) reflects the transfer ability in a limited way, but it is still not enough to use it to predict the transfer learning performance**, as it does not predict which representation is better in transfer learning when we compare multiple of them.

5. **Representations have inconsistent performance regarding the dynamics awareness and linear probing measures.** Thus they are not predictive of transfer performance. In Simple Maze, representations learned with auxiliary tasks that have higher scores than *No auxiliary* representation also show better transfer learning performance, and vice versa (Figure 5.10 and 5.11). However, in Picky Eater, the result is different – for dynamics awareness, representations learned with auxiliary tasks have similar or worse scores than the *No auxiliary* representation, while the transfer learning curves show a different result (Figure 5.21, 5.22, 5.23, and 5.24). For linear probing, most auxiliary tasks do not show a significant difference comparing with the *No auxiliary* task, except those predicting expert information which forces the representation to emphasize this information

during learning.

6. We notice that **all representations show similar performance on non-interference measure** (Figure 5.13 and 5.26). It is possible that non-interference only has a limited effect on transfer learning, this may also suggest that we need to reconsider the formula we use and look for a better measure of non-interference.

7. In two measures related to the value function—**complexity reduction, and specialization—all experiments show that a larger difference between the original task and the transfer task leads to a larger change in the property measure**. This difference comes from the change in the value function, which is caused by changes in transition and reward function (Figure 5.7, 5.8, 5.18, and 5.19). However, these measures are not correlated to the transfer learning performance either. Affected by the small Lipschitz constant, the representation can gain a high complexity reduction score even when it performs poorly in transfer learning.

5.7 Open Questions and Possible Answers

1. **Should I use auxiliary task when learning representation end-to-end?**

   Based on our experimental results, we confirmed that most auxiliary tasks help with improving the transfer performance especially when the difference between transfer task and original task is large. However, when the transfer task is similar to or same as the original task, we did not see obvious advantage given by auxiliary task. In conclusion, if the representation is supposed to generalize to a different task, then training with an auxiliary task, such as predicting states in future or auxiliary control, could be helpful.

2. **What auxiliary task should I use when learning representation for transfer?**

There were 3 auxiliary tasks consistently help with generalizing to a transfer task in both domains—next-agent-state prediction, successor-feature prediction, and auxiliary control. So the prediction on future states and auxiliary control tasks help with generalizations.

We also saw a negative effect when adding input-decoder as auxiliary task, but notice that the input-decoder auxiliary task has been empirically shown effective in literature when it is used to prevent the representation from converging to 0, if 0 is a fixed point of the problem that the agent is trying to deal with. In short, auxiliary task should be chosen according to the goal that the agent would like to achieve.

Chapter 6

Conclusion and Future Work

Our goal was to make progress towards the question: can we better understand the representations learned by deep reinforcement learning agents? Towards this goal, we surveyed an array of ideas on architectures (Chapter 4) and properties (Chapter 3). We designed experiments to assess learned representations, under auxiliary losses, focusing on both when the representation was transferable and not (Chapter 5).

From the mountain of data, we distilled down key comparisons, confirming that auxiliary losses are beneficial for transfer. In particular, adding decoder as auxiliary tasks to require the representation to keep all details—which is unnecessary in learning—hurts the transfer learning speed, and it could learn to solve the task with a much lower speed comparing to others.

Regarding properties, we found that representations with a higher levels of orthogonality typically had a relatively efficient transfer learning, but a good transfer performance did not imply a high orthogonality. Moreover, our experiment suggests that diversity may be limited for predicting transfer performance. When transferring to a different task, the diversity of the no auxiliary representation drops, while the diversity of representations with auxiliary task remains relatively stable. But the current measure takes the risk of being affected by the absolute value of the weight in value function—We will need to improve this measure.

We also found that several measures were not predictive of performance, and instead were skewed by failures of learning, namely complexity reduc-

tion and decorrelation. Not being predictive either, some measures are similar for all representations, such as non-interference. Some measures do not show consistent pattern across domains when compared with the transfer learning task—linear probing accuracy and dynamic awareness. These results also suggested that we need to improve our measures and find new ones that are more robust to the sometimes erratic behaviour of reinforcement learning agents.

This work focuses on understand the representations learned by deep RL agents. We seek to understand the mathematical properties of these representations—the knowledge encoding—of agents that are good and bad at transfer. Much of the previous work in this area has focused on visualizing the agent's knowledge, which is problematic because we leave it to humans to see what they want in the data; to extract correlations that might be biased in different ways. We explicitly try to avoid this bias in two ways: (1) we study agents that are not forced to learn representations that are human interpretable—the agents learn subjective features specific to the training data that typically do not mean much to humans; (2) we do not induce measures or properties related to human-defined notions of how an agent should represent knowledge (in RL and robotics we often call such properties objective or public). As such our work focuses strongly on simply understanding current agents, architectures, and training regimes using a very systematic, mathematical, and performance mindset. Our results should not be used to draw conclusions about how popular deep RL agents are similar and different from humans.

Though we have made conclusions based on the above results, there remain things we would like to investigate further. The first thing is to improve the measure and look for better definitions for properties we have listed in this work, especially the diversity. Futhermore, there has been a substantial effort to characterize transfer, generalization, and overfitting in deep reinforcement learning, primarily in terms of performance [10], [14], [44], [51]—motivating the need for new domains and evaluation methodologies. Notably, prior work illustrated representation transfer is possible across Atari modes [14], but did not yet quantify any properties of those representations. This strongly motivates

another future work: investigating the impact of other algorithmic choices—like regularization—on the properties of emergent representations in a broader suite of domains. The next of which is to understand how the next-agent-state auxiliary task affects the learned representation. In Picky Eater, when measured by both dynamics awareness and linear probing accuracy (color) (Figure 5.21 and 5.23), this auxiliary task posed a negative effect to the learned representations. Though this observation does not affect the conclusion we made, a closer look on its performance may help us to find more about how these properties affect the transfer learning. Last but not least, during this study, we noticed that same properties can be change with time, we are interested in checking measures online, to see how these scores change during learning and to verify whether both the property and the learned policy get stabled simultaneously.

References

[1] G. Alain and Y. Bengio, "Understanding intermediate layers using linear classifier probes," *arXiv preprint arXiv:1610.01644*, 2016. 14

[2] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," in *Advances in Neural Information Processing Systems*, 2019. 14

[3] A. Atrey, K. Clary, and D. Jensen, "Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning," *International Conference on Learning Representations*, 2020. 4

[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. 4, 23

[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, 4, 22

[6] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013. 2, 4, 11, 21

[7] R. G. Brereton, "Orthogonality, uncorrelatedness, and linear independence of vectors," *Journal of Chemometrics*, 2016. 18

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

[9] W. Chung, S. Nath, A. Joseph, and M. White, "Two-timescale networks for nonlinear value function approximation," *International Conference on Learning Representations*, 2019. 22

[10] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *arXiv preprint arXiv:1812.02341*, 2018. 60

[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. 4

[12] 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. 2

[13] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, 1990. 2

[14] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and regularization in dqn," *arXiv preprint arXiv:1810.00123*, 2018. 60

[15] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, Journal of Machine Learning Research, 2017. 3, 4

[16] 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 (ICRA)*, 2016. 4

[17] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau, "Combined reinforcement learning via abstract representations," in *AAAI Conference on Artificial Intelligence*, 2019. 3, 4, 22

[18] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, 1999. 3

[19] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017. 4

[20] 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. 4

[21] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner, "Towards a definition of disentangled representations," *arXiv preprint arXiv:1812.02230*, 2018. 4

[22] 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*, Journal of Machine Learning Research, 2017. 3, 4

[23] 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. 2, 3

[24] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016. 3, 4, 22, 25

[25] K. Javed and M. White, "Meta-learning representations for continual learning," in *Advances in Neural Information Processing Systems*, 2019. 3

[26] P. Kanerva, *Sparse distributed memory*. MIT press, 1988. 2

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 31

[28] 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

[29] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *Twenty-fifth AAAI Conference on Artificial Intelligence*, 2011. 2

[30] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep successor reinforcement learning," *arXiv preprint arXiv:1606.02396*, 2016. 21

[31] 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. 4

[32] T. Lattimore and C. Szepesvari, "Learning with good feature representations in bandits and in rl with a generative model," *arXiv preprint arXiv:1911.07676*, 2019. 2

[33] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for mdps," in *International Symposium on Artificial Intelligence and Mathematics*, 2006. 11

[34] Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling, "State of the art control of atari games using shallow reinforcement learning," *arXiv preprint arXiv:1512.01563*, 2015. 3, 11

[35] B. Lütjens, M. Everett, and J. P. How, "Certified adversarial robustness for deep reinforcement learning," in *Conference on Robot Learning*, 2019. 18

[36] 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, 21, 23

[37] 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

[38] D. J. Mankowitz, A. Žídek, A. Barreto, D. Horgan, M. Hessel, J. Quan, J. Oh, H. van Hasselt, D. Silver, and T. Schaul, "Unicorn: Continual learning with a universal, off-policy agent," *arXiv preprint arXiv:1802.08294*, 2018. 25

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. 3

[40] 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, 8

[41] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," in *Advances in Neural Information Processing Systems*, 2018. 4

[42] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 4

[43] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems*, 2017. 3, 22

[44] 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. 60

[45] C. Painter-Wakefield and R. Parr, "Greedy algorithms for sparse reinforcement learning," *arXiv preprint arXiv:1206.6485*, 2012. 2

[46] 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.," *International Conference on Machine Learning*, 2008. 22

[47] 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

[48] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017. 22

[49] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation*, IEEE, 2018. 4

[50] M. M. Rahman, S. H. Rashid, and M. Hossain, "Implementation of q learning and deep q network for controlling a self balancing robot model," *Robotics and biomimetics*, vol. 5, no. 1, p. 8, 2018. 3

[51] 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. 60

[52] B. Ratitch and D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning," in *European Conference on Machine Learning*, 2004. 2

[53] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing-solving sparse reward tasks from scratch," *arXiv preprint arXiv:1802.10567*, 2018. 25

[54] C. Rupprecht, C. Ibrahim, and C. J. Pal, "Finding and visualizing weaknesses of deep reinforcement learning agents," *CoRR*, 2019. 4

[55] D. Russo and B. Van Roy, "Eluder dimension and the sample complexity of optimistic exploration," in *Advances in Neural Information Processing Systems*, 2013. 2

[56] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016. 4

[57] T. Schaul, D. Borsa, J. Modayil, and R. Pascanu, "Ray interference: A source of plateaus in deep reinforcement learning," *arXiv preprint arXiv:1904.11455*, 2019. 19

[58] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *International*, 2015. 23

[59] 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," *arXiv preprint arXiv:1911.08265*, 2019. 3, 22

[60] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, *et al.*, "The predictron: End-to-end learning and planning," in *International Conference on Machine Learning*, Journal of Machine Learning Research, 2017. 3, 4

[61] J. Sorg and S. Singh, "Linear options," in *International Conference on Autonomous Agents and Multiagent Systems*, 2010. 1

[62] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," *arXiv preprint arXiv:1804.00645*, 2018. 1, 4, 22

[63] K. L. Stachenfeld, M. Botvinick, and S. J. Gershman, "Design principles of the hippocampal cognitive map," in *Advances in Neural Information Processing Systems*, 2014. 13

[64] F. P. Such, V. Madhavan, R. Liu, R. Wang, P. S. Castro, Y. Li, J. Zhi, L. Schubert, M. G. Bellemare, J. Clune, *et al.*, "An atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents," *arXiv preprint arXiv:1812.07069*, 2018. 4

[65] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, 1996. 2

[66] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction 2nd ed.* MIT press Cambridge, 2018. 1, 2, 25

[67] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup, "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," in *International Conference on Autonomous Agents and Multiagent Systems*, 2011. 6

[68] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. P. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," *International Conference on Machine Learning*, 2012. 1

[69] R. S. Sutton and S. D. Whitehead, "Online learning with random representations," in *International Conference on Machine Learning*, 1993. 2

[70] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on Artificial Intelligence and Machine Learning*, 2010. 22

[71] E. Talvitie, "Self-correcting models for model-based reinforcement learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 2

[72] S. Thrun, "Is learning the n-th thing any easier than learning the first?" In *Advances in Neural Information Processing Systems*, 1996. 3

[73] J. Tyo and Z. Lipton, "How transferable are the representations learned by deep q agents?" *arXiv preprint arXiv:2002.10021*, 2020. 4

[74] B. Van Roy and S. Dong, "Comments on the du-kakade-wang-yang lower bounds," *arXiv preprint arXiv:1911.07910*, 2019. 2

[75] Y. Wan, M. Zaheer, A. White, M. White, and R. S. Sutton, "Planning with expectation models," 2019. 2

[76] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, 1992. 7

[77] 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. 4

[78] M. White, "Unifying task specification in reinforcement learning," in *International Conference on Machine Learning*, Journal of Machine Learning Research, 2017. 6

[79] G. Yang, A. Zhang, A. S. Morcos, J. Pineau, P. Abbeel, and R. Calandra, "Plan2vec: Unsupervised representation learning by latent plans," *arXiv preprint arXiv:2005.03648*, 2020. 4, 22

[80] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015. 4

[81] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*, 2016. 4

[82] A. Zhang, N. Ballas, and J. Pineau, "A dissection of overfitting and generalization in continuous reinforcement learning," *arXiv preprint arXiv:1806.07937*, 2018. 4