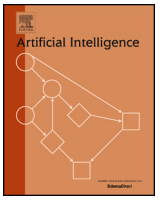
Artificial Intelligence 330 (2024) 104100

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Investigating the properties of neural network representations in reinforcement learning

Han Wang [a,b,*], Erfan Miahi [a,b], Martha White [a,b,c], Marlos C. Machado [a,b,c], Zaheer Abbas [d], Raksha Kumaraswamy [a,b], Vincent Liu [a,b], Adam White [a,b,c]

[a] Department of Computing Science, University of Alberta, Canada
[b] Alberta Machine Intelligence Institute (Amii), Canada
[c] Canada CIFAR AI Chair
[d] Google Deepmind, Canada

ARTICLE INFO

Keywords:
Representation learning
Reinforcement learning
Neural networks
Representation transfer
Auxiliary tasks

ABSTRACT

In this paper we investigate the properties of representations learned by deep reinforcement learning systems. Much of the early work on representations for reinforcement learning focused on designing fixed-basis architectures to achieve properties thought to be desirable, such as orthogonality and sparsity. In contrast, the idea behind deep reinforcement learning methods is that the agent designer should not encode representational properties, but rather that the data stream should determine the properties of the representation—good representations emerge under appropriate training schemes. In this paper we bring these two perspectives together, empirically investigating the properties of representations that support transfer in reinforcement learning. We introduce and measure six representational properties over more than 25,000 agent-task settings. We consider Deep Q-learning agents with different auxiliary losses in a pixel-based navigation environment, with source and transfer tasks corresponding to different goal locations. We develop a method to better understand *why* some representations work better for transfer, through a systematic approach varying task similarity and measuring and correlating representation properties with transfer performance. We demonstrate the generality of the methodology by investigating representations learned by a Rainbow agent that successfully transfers across Atari 2600 game modes.

1. Good representations for RL

In reinforcement learning an agent interacts with their environment, receiving observations and taking actions based on those observations, with the goal of maximizing the sum of a special numerical signal, the reward. The function that converts these observations is known as *representation*, its elements are known as *features*, and the process of learning such a function is known as *representation learning*. Ultimately, many other subproblems depend on the agent's representation. Bad representations hinder predictions and diminish the effectiveness of planning and learning algorithms [81,29,85,14]. Good representations can lead to better sample efficiency [67,37]. Therefore, the key question motivating this research is: *what are good representations and how can the agent find them?*

* Corresponding author.
E-mail address: han8@ualberta.ca (H. Wang).

https://doi.org/10.1016/j.artint.2024.104100
Received 21 February 2023; Received in revised form 17 January 2024; Accepted 19 February 2024

Available online 1 March 2024

Good representations are classically defined in service to other tasks. Often, good representations are said to be those that improve agents in some dimension, such as learning efficiency [87,30], performance in unseen tasks [69,18,4,28,63,93,52,84], the accuracy of learned models for planning [73,24,74,36,21,90], and in the agent's ability to represent the world the way humans do [19,27]. In this context, there are two main approaches for obtaining good representations: using a fixed, expert-designed transformations of the agent's observations, or learning such transformations from data.

Fixed transformations of the agent's observations, which lead to fixed-basis architectures, have been extensively explored in reinforcement learning. They allow us to enforce specific properties that are thought to be beneficial. For example, many approaches either use or search for orthogonal or decorrelated features, such as orthogonal matching pursuit [58], Bellman-error basis functions [61], Fourier basis [35], tile coding [78], and proto-value functions [49,34]. Prototypical input matching methods have been explored, as in kernel methods, radial basis functions [79], cascade correlation networks [15], and Kanerva coding [32]. Some of these approaches produce high-dimensional, sparse representations, which are more likely to be orthogonal. Moreover, by activating only a small subset of features, a sparse representation reduces computation and increases scalability, such as in tile coding [78] and in sparse distributed memories [65]. However, fixed-basis architectures are not adaptive and are difficult to scale to high-dimensional inputs.

Recent developments in representation learning for reinforcement learning explore a different perspective: we should avoid optimizing specific properties[1] and instead use gradient descent to let the training data dictate the properties of the representation. This is achieved with specific training regimes, including multi-task (parallel) training [22,8,83], auxiliary losses [30,5], and training on a distribution of problems (à la meta-learning) [18,73,56,72,31]. The underlying idea is that good representations will emerge if the problem setting is complex enough, where goodness is often measured by success on some held-out test task.

There are many different ways to evaluate and understand these emergent representations. Recent work has explored this question in roughly two ways: what do good representations look like, and what capabilities do good and bad representations allow. The most common approach is to visualize the learned representations [51,87,24,66,92,19,23,77,36,74,12,21,5]. This approach has been used, for example, to provide evidence for the emergence of abstraction and compositionality in supervised learning [91,54]. However, in reinforcement learning, the impact of delayed consequences and temporally correlated data make it difficult to import these analysis techniques from other fields, and recent work highlighted how popular approaches like saliency maps may not always be appropriate [3].

This paper explores the properties of representations learned by deep reinforcement learning systems: specifically (1) DQN [51]—Q-learning with neural network function approximation—combined with different auxiliary tasks [30], and (2) Rainbow [26]. We investigate properties grouped in three categories: *capacity* (complexity reduction, dynamics awareness, and diversity), *redundancy* (orthogonality and sparsity), and *robustness* (non-interference). Our property set consists of both a subset of properties discussed in the literature, and properties newly introduced in this paper. We focus on the fully observable setting; our goal is to understand how the representation transforms the current input. Other properties and experiments would be suitable to understand (recurrent) representations that summarize histories for the partially observable setting, but this is left to future work.

We focus on a *representation transfer* setting: a training phase to learn a representation in one *source task*, followed by testing in a related *transfer task* where the agent learns using that fixed, pre-learned representation. Our primary goal is to identify representations that are useful for future learning. Ideally, it should be useful for future learning in the same task. Additionally, the pre-learned representation should also be useful later when learning about other related tasks. Such a representation should enable faster learning, compared with re-learning everything from scratch, even if we prevent the agent from changing the representation after pre-training—perhaps the clearest and sternest evaluation of the usefulness of prior learning. If fine-tuning or transfer of both the representation and the value function are required to outperform learning from scratch, then one might fairly wonder if any representation is learned at all. In the simple navigation environment we use below, we know there exists a simple two dimensional representation that is sufficient to transfer between navigation tasks in the same maze. In such domains, we postulate that a representation that cannot support successful transfer without fine-tuning is not a useful representation.

Representation transfer, as defined above, relies on a set of related tasks to evaluate future learning. We start with a simple image-based maze environment, that allows exhaustive experiments, as well as provides a natural notion of task similarity (goal location). We then test across different game modes in Atari 2600 games [47]. Recent work has demonstrated that transfer between game modes is possible with a variant of the Rainbow agent [68], however transfer is clearly nontrivial as prior investigations with DQN reported failure [17].

Our first study in the image-based maze environment uses nine auxiliary tasks, resulting in 150 representations for 173 target tasks. We investigated two activations: the widely used ReLU which produces a relatively compact, dense representation, and a new activation function called FTA [59] that produces high-dimensional, sparse representations. The key insights are as follows.

1. Auxiliary tasks can facilitate emergence of representations effective for transfer, however many auxiliary tasks do not outperform learning from scratch (do not transfer) with ReLU networks.
2. Using sparse activations (FTA) was a significant factor in improving transfer. The FTA-based representations transferred consistently, with or without auxiliary tasks.

[1] There is, of course, work in reinforcement learning exploring how to encode specific properties on the network (e.g., sparse activations [42,59], disentangled features [28], and orthogonality constraints [89,21]).

3. ReLU-based representations transfered well to very similar tasks (better than FTA), but significantly worse than FTA to less similar tasks.
4. Transfer was not possible with linear function approximation: performance was significantly better when the representation was inputted into a nonlinear value function.
5. The representations that transferred best had high levels of complexity reduction, medium-high levels of dynamics awareness and diversity, and medium levels of orthogonality and sparsity.

A key contribution of this work is providing a systematic approach to investigate representations and their properties. The empirical design took many iterations, including 1) developing the transfer setup and a way of ranking task similarity using successor features so that we could systematically vary the level of difficulty in transfer, 2) developing the set of properties to measure, 3) appropriately sweeping hyperparameters to obtain reasonably performing agents but still avoiding over-tuning, and 4) providing several mechanisms to aggregate and visualize the mountain of data produced across representations. For example, initial results had little consistency because the agents themselves were not effectively trained; it turns out analyzing poorly performing agents results in unclear conclusions.

Using these insights, we applied our methodology to understand representation transfer across different Atari 2600 game modes [47]. We trained a Rainbow agent on the default game mode, and showed that the learned representation facilitated transfer to other modes in three different games. We found similar outcomes in terms of the properties and transfer performance, which is both somewhat surprising and an indicator that the properties and methodology we propose here are meaningful. More specifically, we similarly found that the representation learned by Rainbow had:

1. high complexity reduction—increasing significantly from its random initialization,
2. high orthogonality and sparsity, even more so than the representations in the maze environment,
3. and medium diversity.

The results in this work complement the growing literature on representation transfer in reinforcement learning by providing a quantitative approach to understand learned representations. We start the paper by more explicitly defining representations (Section 2) and what it means to be a good representation (Section 3). We then explain the auxiliary losses (Section 4) that result in the many different representations we analyze in the results. We then explain the experimental setup (Section 5) and provide our first key result showing transfer performance, successful or not, for these many representations (Section 6). We then introduce the properties (Section 7) and then analyze these properties (Section 8) showing correlations to transfer performance (using 150 representations), the evolution of property values over time, and finally a contrast between final property values for three representations that provided the bulleted conclusions above. We conclude with results in Atari 2600 games (Section 9).

2. Problem formulation and notation

We formalize the agent's interaction as a finite Markov Decision Process (MDP) with a finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, transition function $P : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0,1]$, and a bounded reward function $R : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}$. On each time step, $t = 1, 2, ...$, the agent takes action A_t in state S_t and the environment transitions to state $S_{t+1} \sim P(\cdot|S_t, A_t)$ and emits a reward R_{t+1}. The agent's objective is to find a policy $\pi : \mathcal{S}\times\mathcal{A} \to [0,1]$, that maximizes the expected discounted sum of future rewards, the *return*, $G_t \doteq R_{t+1} + \gamma_{t+1}G_{t+1}$, where $\gamma_{t+1} \in [0,1]$ denotes a discount that depends on the transition (S_t, A_t, S_{t+1}) [88]. In episodic problems, which we study here, the discount might be 1 during the episode, and it becomes zero when S_t, A_t lead to termination.

For much of this work we study the representations learned by DQN [51], a widely used value-based method in deep reinforcement learning (In Section 9, we study Rainbow [26], and leave the description to then). The approximate value function is parameterized by a set of weights θ: $Q_\theta(s,a) \approx \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$. The word *deep*, in deep reinforcement learning, stems from the use of neural networks to approximate the expected return. In particular, DQN iteratively updates its action-value estimates by training the parameters of a neural network, θ, with stochastic gradient descent: $\Delta\theta \propto \left(R_{t+1} + \gamma_{t+1}\max_a \bar{Q}_\theta(S_{t+1}, a) - Q_\theta(S_t, A_t)\right)\nabla_\theta Q_\theta(S_t, A_t)$. The target network, $\bar{Q}_\theta$, is not updated on every step, but only periodically set equal to the current Q_θ. Actions are selected according to an ϵ-greedy policy, where $A_t \doteq \arg\max_{a\in\mathcal{A}} Q_\theta(S_t, a)$ with probability $1-\epsilon$ or a random action with probability ϵ. As is typical, we use mini-batch updates from an experience replay buffer [41].

We use image inputs, a convolutional network and a fully connected layer with a lower dimension d. We consider two different activation functions on this fully connected layer: rectified linear units (ReLU) [53] and fuzzy tiling activation (FTA) [59]. FTA is a one-to-many activation that leads to a larger number of units in the representation layer: $k \times d$ instead of d, but with only a small number of active features at once. This allows us to investigate more compact, lower-dimensional representations produced by the ReLU and higher-dimensional, sparse representations produced by FTA. We call these representations the *representation layer*.

We make extensive use of auxiliary tasks [30] to both induce better representations and to study them. Auxiliary tasks are additional prediction tasks given to agents to incentivize the network to learn about properties of the environment which are, in principle, not directly related to reward maximization. Examples include predicting pixel changes [30] and the next state given an action [55]. These tasks are posed as additional loss functions and the agent is tasked to balance between them and return maximization.

We use a unified architecture to explore representations induced by a variety of different auxiliary tasks, as shown in Fig. 1. The first layers, parameterized by θ_R, produce the representation $\phi_t = \Phi_{\theta_R}(s_t)$. The last layers, which are parameterized with θ_V, use the representation to estimate the action-values. Auxiliary tasks are encoded with additional layers and separate heads (with parameters

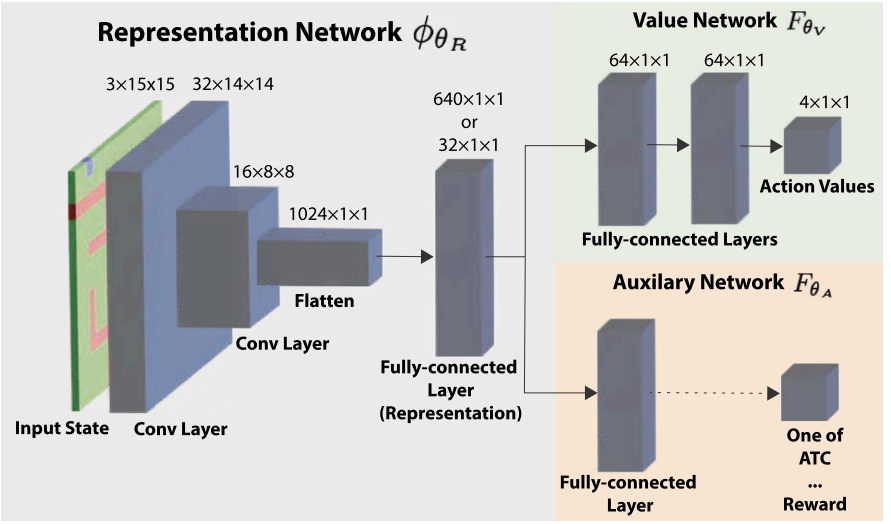


Fig. 1. We experiment with agents using this network architecture, with different auxiliary losses. The representation network, Φ_{θ_R}, learns a mapping from input-state s_t to the agent-state (representation of s_t). The representation network is learned to improve two objectives: performance on a main task and on an auxiliary task. The auxiliary tasks we use are described in Section 4. Our agents only use one auxiliary task at a time.

θ_A), further impacting the updates to θ_R via gradient descent: $\Phi_{\theta_R}(s)$ must be adjusted to be useful for both estimating action-values and reducing the auxiliary losses.

Given this setup, a natural and fundamental question is: *what is the representation in a deep RL agent?* The answer relies on the role of the representation, which is primarily to promote *future learning*. We learn a transformation on the inputs—features—to facilitate downstream learning. The features should 1) be *reusable* or generally useful for multiple predictions, 2) improve *sample efficiency* for the given online algorithm (e.g., SGD), and 3) be *computationally efficient*.[2]

For example, an agent may want to pickup objects in a room, with new objects continually added. The features could describe the objects, so that new objects *reuse* previously learned concepts (e.g., red, cup) that are a succinct (efficient) description of the object. If the agent uses online updating, from temporally correlated samples, we may prefer sparsely activated features that only change a subset of the weights, to reduce interference and promote *sample efficiency*.

Of course, this is only a hypothetical example; we do not truly know the semantics of what is learned, nor what features would improve learning. In this paper, we attempt to systematically measure representation properties and relationships to transfer performance (future learning), to gain more insight into the representations learned by deep RL agents.

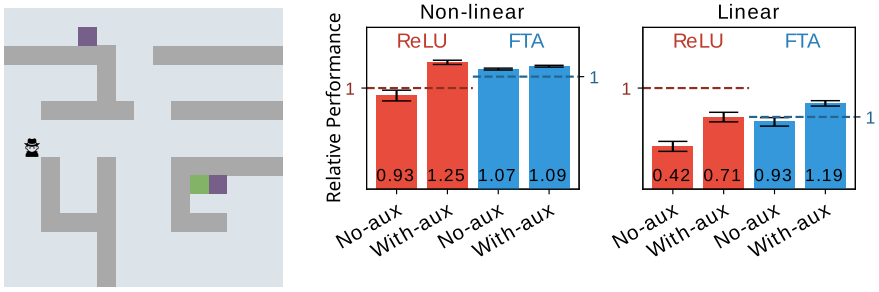
3. Good representations for transfer

We aim to understand the properties of representations that emerge in deep reinforcement learning, but it is critical to do so for both good and bad representations; which, again, begs the question: *what is a good representation*? We use a simple definition: a good representation is one that transfers. More precisely, if features learned during an initial learning phase allow for faster learning on future data, then those features transfer. Good representations that reliably achieve good transfer may exhibit properties and attributes different from representations that result in poor or negative transfer.

We seek to empirically relate representation properties and performance, which requires an environment where transfer is possible. We investigate how the complexity of the value function interacts with transfer in a simple pixel-based navigation environment with obstacles. This environment, depicted in Fig. 2a, can be readily used to generate numerous related tasks. The agent must learn to navigate to a given goal state in as few steps as possible. The problem is episodic, with $\gamma = 0.99$, a reward of $+1$ when reaching the goal and 0 otherwise. The input state consists of an RGB input of a 15×15 grid (size $15 \times 15 \times 3$), encoding the agent's current location (but not the goal). The actions correspond to the four cardinal directions, and transition the agent deterministically by one pixel, or not at all if the action is into a wall. To simplify exploration, the agent starts in a uniform random state and episodes are cut-off at 100 steps; the agent is then teleported to a new random state and this transition is discarded.

We use different navigation tasks (different goal locations) to define training and transfer in our environment. The agent is first trained to go to the goal at a specific location (e.g., $[9, 9]$, depicted in Fig. 2a). Next, we create a new agent with the parameters of the network up to the representation layer copied over from the training agent and we freeze them to prevent further adaption—a transfer agent. In this transfer phase of the experiment, the new agent is trained to navigate to a nearby but different goal location.

[2] Another criteria that has been considered for partially observable settings, especially where the true state is low-dimensional and compact, is how well the representation captures the true state, with related ideas about finding a compact set of disentangled features or causal features; see the work by [39] for a nice overview. Such representations could facilitate all three of these goals—reusability, sample efficiency and computational efficiency—but may not be necessary to achieve them.



(a) Maze problem (b) Relative Performance

Fig. 2. Representation transfer is possible in the Maze navigation task and auxiliary tasks improve transfer when using non-linear function approximation. (a) The position of the walls (dark gray), the goal (green) in the training, and two transfer tasks (purple) are shown. (b) Performance relative to the baseline (from scratch) agent: above 1 represents an improvement and below 1 denotes negative transfer. Error bars show a 95% confidence interval. Performance is reported for the best auxiliary task for each activation: VirtualVF5 for ReLU-based representations, and SF for FTA, explained in Section 4. The dotted lines labeled with a 1 are at different locations, to indicate the relative performance between FTA and ReLU from scratch. In non-linear, FTA from scratch was better; in linear, ReLU from scratch was better. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

We compare this to a baseline agent where all parameters of the network are not pre-trained but instead randomly initialized and updated during the transfer phase. This baseline agent learns the task from *scratch* because it does not benefit in any way from prior learning. If the transfer agent learns more quickly than the baseline scratch agent, then we say the representation learned in the first phase of the experiment *facilitates transfer*. Because of the multiple possible goal locations, there are many possible training and transfer tasks in this environment.

At this point we have the major ideas in place to present a simple but foundational result: *representation transfer is possible in our navigation task and auxiliary tasks improve transfer*. Fig. 2b summarizes this result upfront; we give details later about the experimental setup as we proceed to more complex and nuanced results. Representations pre-trained on the training task significantly outperformed representations learned from scratch on the transfer task. In addition, we found auxiliary tasks were important for transfer, at least for ReLU. Pre-trained representations using ReLU exhibited negative transfer, whereas ReLU-based representations combined with well-designed auxiliary tasks did transfer.

These results, though, produced some surprises. First, even in this simple environment, transfer with ReLU and auxiliary tasks was difficult. This reaffirms some of the previous anecdotal and documented [17,84] issues with transferring representations. In fact, switching to a different (sparse) activation, FTA, had a bigger impact than any auxiliary task. This suggests some issues with the representations learned under ReLU.

Second, *we were unable to obtain successful transfer with linear value functions*. This is achieved by removing the two fully connected layers in Fig. 1. This outcome was not for a lack of effort. Fig. 2b strongly suggests that, in our navigation environment, non-linear value functions significantly improve transfer, even in a relatively simple environment. We see that when the value function was linear in the features, neither a pre-trained representation (labeled ReLU and FTA) nor any other representations trained with auxiliary tasks improved over training a fresh representation from scratch. This suggests representations may emerge in earlier layers of the network, and that it may be more feasible to learn re-usable features when they can be nonlinearly combined, even if only with a simple shallow network. We highlight this important point here; for the remainder of this paper, we restrict our scope only to non-linear value functions.

We presented these overall results upfront, before diving deeper and understanding *why* we see these outcomes. In the next sections, we describe the auxiliary tasks we used to find good representations; then how we evaluate the representations for different transfer tasks; and finally give insights into failures in transfer under ReLU due to transfer task similarity. Later, we dive even deeper, measuring and correlating representation properties to transfer performance.

4. The auxiliary losses

We focus on the representations that emerge under different auxiliary losses for two reasons: 1) Adding or changing auxiliary tasks does not change the functional capacity and size of the learned representations; the consistency on the capacity makes comparisons more interpretable. 2) The role and impact of auxiliary losses in reinforcement learning remains poorly understood, and constitutes an important area of study. We introduce the general idea behind each auxiliary task in text, while their formal definition is presented in Table B.1 in Appendix B.1.

Input Reconstruction (IR): This auxiliary task tries to reconstruct the network's input, as in an autoencoder [50]. This extraction is achieved by using a bottleneck layer: a low-dimensional layer that forces only the most important information to be retained and the remainder, including the noise, to be discarded. We include this auxiliary task as a classic and simple choice.

Next Agent State Prediction (NAS): Another common choice is to predict the next agent state [30,62,9,48,21,90,74,56,72]. This loss encourages the representation to capture the transition dynamics. The agent predicts ϕ_{t+1} using ϕ_t and action a_t. Predicting the next agent state might give vacuous solutions when it is the only training signal because the neural network may converge to the trivial solution, such as assigning zero as the representation to all states [21,82]; jointly training with the main task, however,

prevents this from happening. The combination of this auxiliary loss with the main task encourages the representation to both be useful for action-value estimation, as well as capable of anticipating features on the next step. Several papers have highlighted that the ability to predict the next state is related to the ability to predict action-values [5,60,80].

Successor Feature Prediction (SF): NAS can be taken one step further, with the target including not just the next agent-state but many future agent states [46]. *Successor features* (SFs) provide just such a target. SFs are defined with respect to a particular policy π as $\psi_t^\pi = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i \phi_{t+i}\right]$. They have been used in the transfer setting because they can be used to quickly infer value estimates for new reward functions that are a linear function ϕ_t [4]. In the tabular case, SFs correspond to successor representations [13], which have an equivalence to proto-value functions [48]. We opt to use the greedy policy according to the action values for the main task, which means the SFs are tracking a changing policy.

Reward Prediction (Reward): Another auxiliary task we consider is predicting the immediate reward in the future based on the current state and action [30]. The prediction requires the agent to encode the reward information that it can obtain in the short term in the representation function.

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

Virtual Value Function Learning (VirtualVF): This auxiliary task is based on the tasks the agent will face in transfer. We consider one auxiliary loss that uses a goal location at the center of the maze (VirtualVF-1), and another that uses five goals at the four corners and the center of the maze (VirtualVF-5). These are virtual tasks, because the agent imagines achieving these goals, even though they are not the training goal. We use VirtualVF-1 and VirtualVF-5 to assess the utility of having a larger set of virtual goals. We learn these auxiliary value functions with DQN.

Augmented Temporal Contrast (ATC): The contrastive loss encourages the network to learn similar representations for input-states that are temporally close to each other [76,1]. This auxiliary task led to the first successful pre-training of a deep reinforcement learning agent, meaning it led to representations that could be generally reused for other tasks. ATC also includes other augmentations, like data augmentation (e.g., randomly flip, crop, rotate or translate the input images). See detailed experiment settings in Appendix B.1. We test it with these additions, to report performance of the originally proposed approach, even though it goes beyond strictly only adding an auxiliary loss.

5. Experiment design

Our experiments consist of two stages: a representation learning stage in a training task, and a transfer stage using this learned representation in a transfer task. In this section, we outline the details for these two stages in our experiments, and outline the agents and how we evaluate them.

5.1. Transfer setup

The first stage is to train the representation. All representations are trained with a DQN agent in the training task, with goal location depicted in Fig. 2a. To prevent overfitting, we employ an *early-saving* strategy to save the representation function, Φ_{θ_R}, as soon as the agent is able to finish 100 consecutive episodes in 100 steps or less. Each representation corresponds to a choice of activation function and auxiliary loss—including choosing not to use an auxiliary loss.

In the second stage, we learn with the representation from the first stage, in a new transfer task. Specifically, we 1) load and freeze the learned representation, 2) re-initialize the value function, and 3) learn the value function for the transfer task with DQN using the fixed representation. No auxiliary tasks are used in transfer, and only the 64×64 value function network is learned with DQN. Learning with a re-initialized value function rather than fine-tuning prevents negative effects from the old value function during transfer, especially to less similar transfer tasks. Further, re-initializing the value function ensures that the difference between transfer learning and learning from scratch is due to the learned representation. The agent learns in this new task for 100,000 steps.

We consider 173 transfer tasks—all possible goal locations, including the training goal state. To sort performance amongst these tasks, we provide a novel method to measure their similarity to the training task. In this way, we can ask questions about transfer to more or less similar transfer tasks. The key idea is to first obtain successor representations for each state, and then compute similarity in this new space. The successor representation encodes similarity based on transition dynamics, meaning that states are considered nearby due to the ability to reach them rather than due to other distances, such as Euclidean distances which do not respect the walls in the Maze. For specific details, see Appendix B.3.

5.2. Network choices and activation functions

To obtain representations with different properties, we use two different activation functions: Rectified Linear Unit (**ReLU**) [53] and Fuzzy Tiling Activation (**FTA**) [59]. We use ReLU in our experiments because it is the most widely used activation function in deep reinforcement learning. FTA is a new activation function that projects the input onto a higher dimension with guaranteed and controllable sparsity. With the two different activation functions, we are able to compare representations with different dimensionality and sparsity levels.

ReLU is a one-to-one activation function, projecting the input z to the same dimensional space, with clipping all negative value to zero. The function is defined as $h_j(z) = \max(z, 0)$ for input z, where z is a linear weighting on the previous layer.

FTA is designed to generate sparse outputs. Essentially, it bins the scalar input into k bins, with some smoothing to ensure non-zero gradients through the activation. The smoothness and bin width is controlled by a parameter, $\eta > 0$. The interval is from $[-\frac{\eta}{2}k, \frac{\eta}{2}k]$, with k equally sized bins of size η. Assume the input z is in bin i, namely $-\frac{\eta}{2}k + (i-1)\eta \leq z \leq -\frac{\eta}{2}k + i\eta$ where $i \in \{1, 2, \ldots, k\}$. The k-dimensional output vector $\mathbf{h}(z)$ given by FTA on z has entries $h_j(z) \in [0, 1]$ defined as

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

Larger η activates more entries in $\mathbf{h}(z)$, and smaller η results in more sparsity. This formulation removes a hyperparameter by using the suggested default choice of $\eta = \delta$.

For our experiments, the representation function consists of two convolutional layers, one linear transformation, and a choice of activation function. The linear layer projects the output of the convolutional layer to a 32-dimensional space. When using ReLU, the representation layer has $d = 32$ features. If FTA is used, it has 640 features since FTA projects each scalar to a short, sparse vector with 20 bins. Note that FTA still uses the same number of learned parameters to produce these 640 features ReLU uses to produce the 32 features, because binning occurs after the linear weighting. However, the outputted number of features is higher, and so the value function and auxiliary tasks all have more parameters, at least in their first layer. We therefore also evaluate ReLU(L)—L for large—which uses 640 features. ReLU(L) uses significantly more parameters than FTA to produce these 640 features.

The structure for the value function and auxiliary tasks is given in Fig. 1. We use two hidden layers with 64 nodes each for the value function, and one hidden layer with 64 nodes for the auxiliary task. We use a simpler network for the auxiliary task to force the representation to learn as much as possible. We use a slightly larger network for the value function, to avoid overly constraining it and so confounding transfer performance.

5.3. Agent specification

We use standard choices for DQN, including the use of ϵ-greedy exploration, an experience replay buffer, target networks, and the Adam optimizer [33]. In total there are 9 choices for auxiliary tasks: No-aux, ATC, IR, NAS, SF, Reward, XY, VirtualVF-1, and VirtualVF-5. There are 3 activation functions: FTA, ReLU, and ReLU(L). When using FTA with auxiliary tasks, we set the number of bins $k = 20$ and $\eta = 0.2$. This implicitly specifies the range for binning to $[-2, 2]$. For the No-aux task agents, we test $\eta = 0.2, 0.4, 0.6,$ and 0.8 and report performance for each, not the best one. This gives a total of 30 agent specifications.

We consider three baseline agents, which we call RANDOM, INPUT, and SCRATCH. They allow us to falsify different hypotheses about the role of the learned representation. RANDOM uses a randomly initialized network as the representation, without any learning. The agents start with a random network, so this baseline checks whether learning actually improved the representation. INPUT omits the representation network and directly inputs the agent's observation to the value function component. It is meant to check if the learned representations play any (useful) role, and if learning from scratch in the transfer task might just have been faster with smaller networks. Finally, SCRATCH is a DQN agent that starts learning from randomly initialized weights in the transfer task. The purpose of learning the representation is to learn faster than learning from scratch in the transfer task. This is the most important baseline, as it defines whether a learned representation was successful for transfer—facilitated learning faster than SCRATCH. It allows us to check whether the learned representation provides shared information between the original and transfer tasks. If so, agents that reuse learned representations should be more data efficient than SCRATCH.

5.4. Reporting performance and hyperparameters

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

Step-size controls the rate of learning in a reinforcement learning agent and can largely affect the agent's performance in practice. The only hyperparameter common across all agents is the step-size. To ensure each agent is evaluated under the same number of parameter configurations, we only swept the step-size. We separately pick the step-size in Stage 1, in the training task, and in Stage 2, when just learning the value function. We use the average performance over 5 runs to select the step-sizes. Specifically, in Stage 1 we run each of the 30 agent specifications (different auxiliary tasks, different activation functions, different parameter settings of the activation function when using FTA, and different representation sizes) with five different step-sizes, 5 times with different random seeds: $30 \times 5 \times 5$ experiments. We select the best step-size according to training AUC, and use the representations produced under those step-sizes. Then in Stage 2, we evaluate each step-size only for those representations, and pick the best step-size for an agent specification for each transfer task by using the average performance across the 5 runs. We sweep the step-size to ensure we are evaluating reasonably well-optimized agents. Additional hyperparameter details, including the selected values, are available in Appendix B.

When we report performance across agents, we do not average across these 5 runs. Instead, each run produces a different representation and we report performance for each one as an independent data point. When showing aggregate performance, we

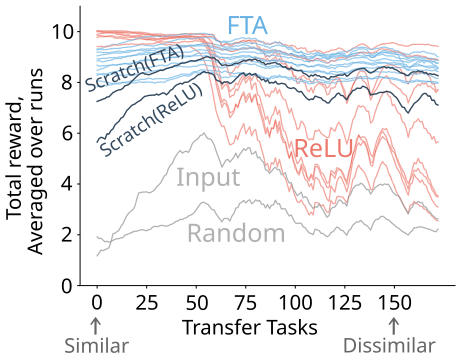


Fig. 3. Transfer performance of 105 different representations (ReLU and FTA) on 173 transfer tasks. The tasks on the x-axis are arranged by similarity to training tasks: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. The dark blue line (SCRATCH) shows the performance when learning in each transfer task from scratch. Lines completely above the SCRATCH line indicate a representation yielded successful transfer in all tasks. Lines that start above the SCRATCH line but fall below as we move left to right indicate a representation that transfers to similar tasks but not to dissimilar tasks. The INPUT and RANDOM baselines are not competitive; for completeness, we still report their performance, but with lighter lines. Overall, many representations achieve transfer and generally FTA-based representations are better on this problem. Details on how task similarity was computed and how this plot was generated can be found in the Appendix. The transfer performance of ReLU(L) is shown in Fig. C.13; it exhibits the same pattern as ReLU, transferring well to similar tasks and not as well in less similar tasks.

aggregate from this larger pool of 5 runs for each of the 30 agent specifications, namely over 150 representations. We do so because each representation has different properties; when correlating agent properties and performance, we may not care which auxiliary task was used, but rather only care about its emergent properties. Averaging across runs compares methods (agent specification) rather than representations.

Finally, we obtain transfer performance in 173 transfer tasks. This means we get 173 transfer performance samples for each of the 150 representations. In total, when aggregating across transfer tasks or agent specifications, we obtain a large number of samples to estimate aggregate performance, even though each agent specification only has 5 runs in the training task. For example, in Fig. 2b, each bar in the plot is for one agent specification and uses $173 \times 5 = 865$ samples to estimate medians, means, and standard deviations. In total, we generate $173 \times 150 = 25,950$ agent-task combinations.

6. Good, bad and ugly representations

We expect some agent specifications to result in representations that aid transfer, and others to impede transfer. UNREAL [30], the first large-scale deep reinforcement learning system to highlight the utility of auxiliary tasks, showed that although auxiliary tasks like pixel prediction improved performance substantially, other tasks such as feature control had a much smaller impact. Other work has highlighted that it can be difficult to obtain any transfer in reinforcement learning [17,84]. It seems the design and deployment of auxiliary tasks remains largely an art.

In this section, we provide some clarity on these discrepancies by showing that 1) there is large variability in performance across auxiliary tasks, and 2) transfer performance can degrade significantly as tasks become more dissimilar.

Fig. 3 summarizes the transfer performance of many different representations corresponding to different auxiliary tasks and activation functions. The plot has task similarity on the x-axis, and each point on the plot summarizes the performance of one representation on one particular transfer task. The lines show how much transfer performance degrades as tasks become more dissimilar.[3] The bold dark blue line shows performance in the transfer task if the representation and value function were trained from scratch—no transfer. Any point above the bold dark blue line indicates a representation that achieved better performance than training from scratch on that task—successful transfer. Any line completely above its corresponding dark blue line indicates a representation that achieved successful transfer for all goal states.

The most important conclusions from Fig. 3 are that 1) several representations achieve successful transfer across all tasks, and 2) a great variety of representations emerge with transfer performance ranging from good to significantly worse than scratch. Looking more closely, some representations achieve successful transfer in dozens of tasks which are most similar to the training tasks, but for tasks less similar to the training one, performance is poor as seen by the step down in many of the lines.

To evaluate the significance of the results in Fig. 3 we must conduct a statistical test. We will focus on dissimilar tasks—as tasks ranked later than 50—because this is where significant differences in transfer ability are most relevant. With 5 runs for one agent specification (activation and auxiliary task pair), we obtain 615 runs in dissimilar tasks in total converting the learning curve into a scalar value by simply computing the area under the curve for each line in Fig. 3. We then ran a *sign test* with a null hypothesis that ReLU-based representation is not worse than scratch representation in dissimilar transfer tasks. The hypothesis is rejected when $p < 0.05$. For many of the ReLU-based representations, they are significantly worse and the null can be rejected. For example,

[3] Fig. 2a shows two transfer tasks as purple squares. The one beside the training goal is most similar according to our ranking, and the other is least similar to the training goal.

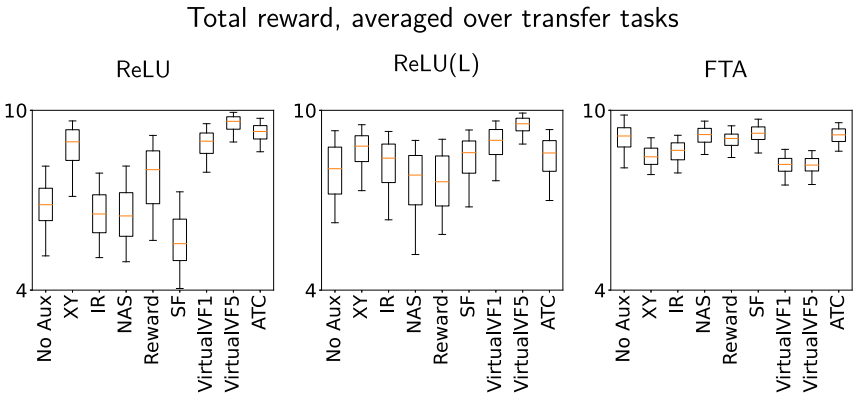


Fig. 4. Transfer performance depends on the activation function, representation size, and auxiliary tasks. Overall, FTA-based representations achieved the best performance and exhibited the least variation in performance across auxiliary tasks. The orange lines depict the median, the upper and lower edges of the box show the 25_{th} and the 75_{th} percentiles, while the whiskers show 1.5 times the inter-quartile range. These results are computed over $173 \times 5 = 865$ samples, and so the standard errors are quite small (as you can see in Fig. C.14 in Appendix C.3).

ReLU+SF was worse than Scratch in 572/615 runs, thus this representation was significantly worse with $p = 2 \times 10^{-119}$. ReLU+NAS was worse than Scratch in 486/615 runs with $p = 6 \times 10^{-50}$. ReLU+Decoder was worse than Scratch in 478/615 runs, given a significant worse performance with $p = 2 \times 10^{-45}$. ReLU+NoAux had 450/615 runs worse than Scratch, resulting in significance with $p = 9 \times 10^{-32}$. On the positive side, the best ReLU-based representation and FTA with no auxiliary tasks were significantly better than scratch. ReLU+VirtualVF5 was better than scratch in 612/615 runs, with $p = 3 \times 10^{-178}$. FTA with no auxiliary tasks performed better than scratch in 547/615 runs, giving a significant better performance with $p = 8 \times 10^{-77}$.

Generally, we found that FTA-based representations yield better representations for transfer compared to ReLU. Almost all FTA representations outperformed Scratch (FTA), and transfer to less similar tasks was effectively the same as on similar tasks to training, as evidenced by the nearly flat lines in Fig. 3. Interestingly, many ReLU-based representations achieved very good performance in transfer to similar tasks but performed significantly worse than Scratch (ReLU)—nearly as bad as the input baseline—on less similar tasks. ReLU (L) performed similarly to ReLU; this result is in Appendix C.2. Another note of interest is that Scratch (FTA) performs better than Scratch (ReLU), and yet FTA representations were still better able to transfer than ReLU-based ones. Using FTA improved on using ReLU; training the FTA representation first in a training task improved performance in the transfer task even more.

Digging a little deeper, Fig. 4 depicts the transfer performance of each auxiliary task. We again see that FTA-based representations achieve higher performance overall and higher performance across auxiliary tasks—the worst performing representation never used FTA, always ReLU. Inspecting each auxiliary task, the FTA-based representations exhibited lower variance across runs. Larger ReLU representations, ReLU(L), did improve performance over the smaller ReLU representations, but not uniformly. The IR auxiliary task representation, for example, improves with large ReLU networks, but ATC performs worse—though not significantly in either case.

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

We included ATC, a recent representation learning strategy, to calibrate the quality of transfer performance. This approach uses multiple networks to compute a contrastive loss, while also using data augmentation of the input images. ATC worked well, but it did not significantly outperform the best ReLU and FTA representations. In addition, we found that the ReLU network combined with data augmentation (random shifting of the input imagines) and no contrastive setup achieved similar performance to ReLU ATC.

7. Representational properties, old and new

Before the emergence of deep reinforcement learning, the study of representations in reinforcement learning and their effects on learning was focused on fixed bases. The problem with this is not that a fixed basis cannot capture complex non-linear relationships (e.g., see the work by [40]), but rather that the representation is fixed—the features are not adapted to the task. In some sense, this is good 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 have proposed and debated numerous properties. We leverage these discussions to analyze our learned representations.

We characterize a representation into three main axes: *capacity*, *efficiency*, and *robustness*. Capacity reflects whether a representation can represent a given function. Efficiency captures the lack of redundancy of the features and the computational cost of using

them. Robustness captures the idea that interference is undesirable and that representations should avoid it; a more complete name is *update robustness*. We define six metrics that capture these three axes and we use them to evaluate the representations learned by our agents. Our goal is to develop a systematic methodology for assessing learned representations, based on a diverse set of properties. This evaluation list does not suggest that a property is necessary; rather, it provides some quantitative measures to supplement more qualitative evaluations like visualizing the representation. Such a list is necessarily incomplete; we attempt only to start with a reasonably broad set of properties.

We assume we have a dataset of 1,000 transitions to measure the properties, $\mathcal{D}_{\text{test}} = \{(s_1, a_1, s'_1, r_1), (s_2, a_2, s'_2, r_2), \ \ldots, (s_N, a_N, s'_N, r_N)\}$, where s'_i is the outcome state from s_i. This dataset is obtained by running the random policy for N episodes with random start states, and then randomly subsampling N transitions, to ensure we cover the state space. We store the transitions because some of the properties rely on consecutive states or the entire transition. The symbol ϕ_i refers to the representation of s_i; $V(\phi) = \max_a Q(\phi_i, a)$ is the value learned given that representation. We compute distances both according to the representation and according to action-values,

$$\begin{aligned} d_{v,i,j} &\stackrel{\text{def}}{=} |V(\phi_i) - V(\phi_j)|, \\ d_{s,i,j} &\stackrel{\text{def}}{=} \|\phi_i - \phi_j\|_2. \end{aligned} \tag{1}$$

7.1. Capacity: retaining relevant information and nonlinear transformations

The first property to consider for a representation is its **capacity**: can it represent the functions we want to learn? The value function network should be a simple function of these features, such as a simple neural network. To measure capacity, we use one direct measure, *complexity reduction*, and two indirect measures, *dynamics-awareness* and *diversity*.

Complexity reduction reflects how much the representation facilitates simplicity of the learned value function on top of those features. If complexity is small, the features encode much of the non-linearity needed. A well-known result is that the composition of a Lipschitz function V with another function ϕ has (Rademacher) complexity: complexity$(V \circ \phi) \leq L$ complexity(ϕ), for L the Lipschitz constant of V. The Lipschitz constant L is one where $\frac{d_{v,i,j}}{d_{s,i,j}} \leq L$ for any ϕ_i, ϕ_j. For a smaller L, the representation ϕ handles most of the complexity, which is good because we have a longer initial learning phase to obtain ϕ. It should be fast to learn V on top of ϕ. Lipschitz value functions have also been motivated for value transfer [38] and model learning [16].

Let us turn this intuition into a measure of how much the representation helps reduce the required complexity in the value function. We can measure $\frac{d_{v,i,j}}{d_{s,i,j}}$ for all pairs (i, j) in our dataset. An estimate of L is the maximum over these slopes. However, empirically we find that the average of these slopes results in bigger differences between representations with better correlation to transfer performance. The max loses significant precision, possibly making very different Q have similar measures; the average summarizes the whole surface. For example, one representation might reduce complexity for most of the space, but due to one small part, have a similarly high maximum slope to a representation that barely reduces complexity in any part of the space. We call these averaged ratios L_{rep}, giving

$$\begin{aligned} \text{Complexity Reduction} &\stackrel{\text{def}}{=} 1 - \frac{L_{\text{rep}}}{L_{\max}} \\ \text{where} \quad L_{\text{rep}} &\stackrel{\text{def}}{=} \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{d_{v,i,j}}{d_{s,i,j}}. \end{aligned} \tag{2}$$

When this ratio is computed on given time step t—either during learning or on the last time step before the representation is frozen for transfer—we use the current action-values. We normalize L_{rep} between 0 and 1 using $L_{\max}$, computed as the maximum L_{rep} over all representations across all time steps. This is subtracted from 1 to ensure higher values refer to higher reduction in complexity.

We can also indirectly measure complexity—that is 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 in fact related to the Laplacian used for proto-value functions [49] and successor features [75,48]. For every state in the dataset, we take its successor state and a random state. If the distance, in representation space, between the successor state is smaller than the distance to a random state, then we say the representation has high dynamics awareness.

$$\text{Dynamics Awareness} \stackrel{\text{def}}{=} \frac{\sum_i^N ||\phi_i - \phi_{j \sim U(1,N)}|| - \sum_i^N ||\phi_i - \phi'_i||}{\sum_i^N ||\phi_i - \phi_{j \sim U(1,N)}||}. \tag{3}$$

In addition, we measure the **diversity** of a representation, which is the opposite of *specialization*. If a representation is specialized to one value function, then it likely uses a small subspace of the larger Euclidean space and likely does not produce a diversity of possible feature vectors. This specialization may be problematic, as it means the representation is unlikely to perform well when it is transferred and used to learn another value function.

To define diversity, we use the ratio between state and value differences. Given two states s_i and s_j, we compare the distance between their representations ($d_{s,i,j}$) and the distance between their values ($d_{v,i,j}$). If the value distance is high—the two state values are very different—then the representation distance is also likely to be high to allow this. The interesting case is when the value distance is low. The representation distance can be high or low, and still allow two states to have similar values, because we project

from a higher-dimensional feature vector to a scalar value. A representation with high diversity would have high representation distance when possible, allowing two states to be distinguished even when they have similar values. A representation with low diversity would simply map these two states with similar values to similar representations, specializing to this value function. The measure is

$$\text{Diversity} \stackrel{\text{def}}{=} 1 - \frac{1}{N(N-1)} \sum_{i,j,i\neq j}^{N} \min\left(\frac{d_{v,i,j} / \max_{i,j} d_{v,i,j}}{d_{s,i,j} / \max_{i,j} d_{s,i,j} + \varepsilon}, 1 \right). \tag{4}$$

A small number ε is added to avoid numerical issues. We normalize by the maximum distances to be invariant to value and representation scales. Diversity can be seen as 1-specialization. The specialization is lower when $d_{v,i,j}$ is small and $d_{s,i,j}$ is large, causing this ratio to be closer to zero. The specialization is higher when the ratio between $d_{v,i,j}$ and $d_{s,i,j}$ is nearly one. Diversity allows us to indirectly measure capacity, as we can check the level of specialization for a given function without needing to have access to the larger set of possible functions.

7.2. Efficiency: feature redundancy

Many function classes can satisfy these capacity properties and so we consider other functional properties of the features. Reducing redundancy in the representation, finding *linearly independent* features, is a basic requirement for improving the representation efficiency. **Orthogonality** satisfies this requirement and additionally provides distributed features as well as minimal interference. For example, factor analysis finds a dense set of orthogonal (latent) factors to explain the data. This representation is highly distributed, as each feature is used to describe many different inputs. At the same time, interference is reduced: the interference for two states with orthogonal feature vectors is zero under linear updating. As before, we normalize magnitudes and ensure higher orthogonality means that more feature vectors, ϕ_i and ϕ_j, are orthogonal to each other.

$$\text{Orthogonality} \stackrel{\text{def}}{=} 1 - \frac{2}{N(N-1)} \sum_{i,j,i<j}^{N} \frac{|\langle \phi_i, \phi_j \rangle|}{\|\phi_i\|_2 \|\phi_j\|_2}. \tag{5}$$

Note that there is an equivalence between orthogonal feature vectors—orthogonal representations—and orthogonal features: the sum over all states i, j of $\langle \phi_i, \phi_j \rangle^2$ is equal to the sum over all pairs of features of the dot product between the vector of those feature values across states (see Appendix A.1). Additionally, for centered features, orthogonality is also equivalent to decorrelation (see Appendix A.3).

One idea related to orthogonality is **sparsity**. If only a small number of features are active for an input, then the features are sparse—with typically the additional condition that each feature is active for some inputs (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. Sparsity has the additional benefit, though, of improving efficiency for querying and updating the function, because only a small number of features are active. To measure sparsity, we calculate, on average, the percentage of inactive features across states in the dataset.

$$\text{Sparsity} \stackrel{\text{def}}{=} \frac{1}{dN} \sum_{i=1}^{N} \sum_{j=1}^{d} \mathbb{1}(\phi_{i,j} = 0), \tag{6}$$

where the function $\mathbb{1}(\cdot)$ is an indicator function, and the representation ϕ_i for state s_i is d dimensional. Equality with zero is tested within a tolerance of 10^{-10}.

7.3. Robustness: interference reduction

More recent work in neural networks has also focused on robustness, both to interference and noise [44,45,2,43,70]. **Interference** reflects how much updates in one state reduce accuracy in other states. We use a recent measure developed for reinforcement learning [43], which uses the difference in temporal difference errors before and after an update. We do the comparison each time the target network is synchronized, which occurs every 64 steps, for a total of T times during learning. For every $t = 1, \ldots, T$, we compare the error between θ_t and the parameters after 64 updates, θ_{t+1}; note t here refers to the synchronization iterator rather than time.

$$\text{Non-interference} \stackrel{\text{def}}{=} 1 - \frac{\text{Interference}}{\text{MaxInterference}}, \tag{7}$$

$$\text{Interference} \stackrel{\text{def}}{=} \{\text{Update Interference}_{t, \geq Percentile_{0.9}}\}_{t=1}^{T},$$

$$\text{Update Interference}_t \stackrel{\text{def}}{=} \frac{1}{N} \sum_{i=1}^{N} \text{err}_{t,i}(\theta_{t+1}) - \text{err}_{t,i}(\theta_t),$$

$$\text{err}_{t,i}(\theta) \stackrel{\text{def}}{=} \left(r_i + \gamma_t \max_a Q_{\theta_t}(s'_i, a) - Q_\theta(s_i, a_i) \right)^2.$$

The maximal Interference is computed across all representations.

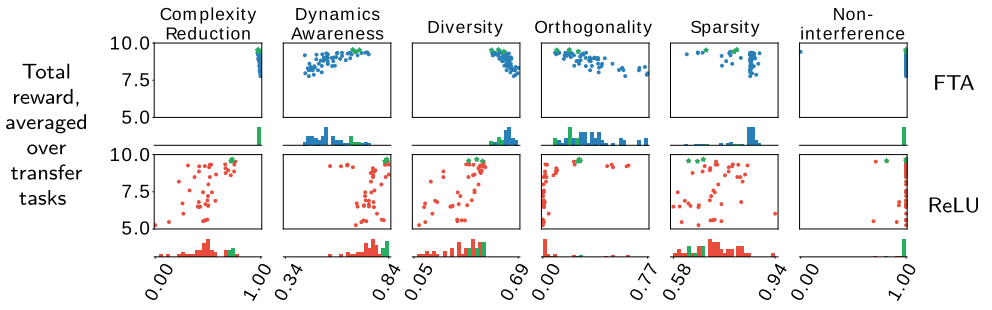


Fig. 5. Performance averaged over transfer tasks versus representation property values. Each dot in a plot corresponds to one representation, with the (x,y) point corresponding to its property value and average transfer performance. The histograms under the plots depict the density of points at each property value. The green stars correspond to the three best performing representations. We separate out FTA-based and ReLU-based representations, which exhibit notably different behavior.

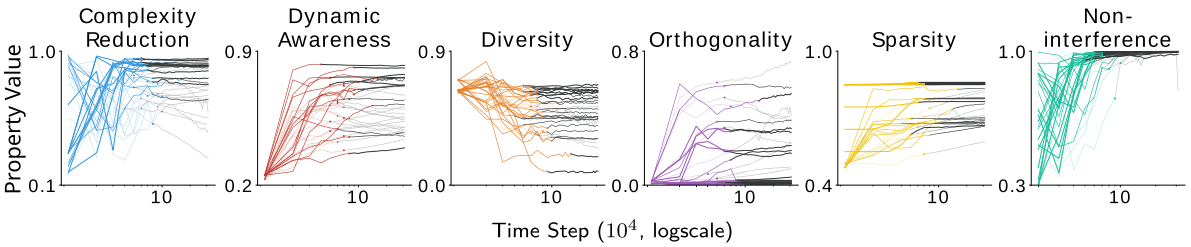


Fig. 6. Plotting the representation properties over time, with one subplot per property. Each curve shows the property of one agent specification (activation and auxiliary task pair), averaged over the 5 runs in the training task. The curve changes color, to black, at the moment in which we took the representation and fixed it; this point was chosen based on when the return for the agent stopped changing. Line colors vary from light gray to black based on how much their values fluctuate from the property value after their return converges, with darker lines denoting lower variation. In these plots, we allowed the representation to keep learning to understand if properties significantly change afterwards. Our primary focus is to show the general trend that properties converge over time, and that they converge approximately when the return does; therefore we use the same color for all agent specifications.

8. The properties of good representations

We can now return to the main question of this work: do good representations that facilitate transfer exhibit particular properties? In this section, we investigate how the properties defined in the previous section relate to transfer performance.

Fig. 5 contains the main result; for now let us focus on only the FTA representations in the top row to better understand this figure. Each subplot shows the transfer performance of every single FTA-representation averaged over all transfer tasks. The representations in each subplot are ordered based on a single measured property. For example, the first subplot in the first row plots the transfer performance of FTA representations and the dots are ordered by *complexity reduction* on the x-axis. Representations with high complexity reduction and good transfer performance would appear as a dot in the top right of the subplot. Representations with low complexity reduction and good transfer performance would appear as a dot in the top left of the subplot, and so on.

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

Now consider the properties of the top performing representations. Again, let us focus our attention on the FTA representations in the top row of Fig. 5. The green stars in each subplot correspond to the top performing representations (in terms of transfer). First notice the stars are typically close together in x and y indicating all three achieve similar performance with similar property values; this is true for ReLU representations as well. In general, we see that the best performing representations are not at the extremes of any property (high or low). Given that FTA representations by and large exhibit high complexity reduction, diversity and sparsity, it is notable that the best performing representations are the lowest of those three properties.

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

The property values depicted in Fig. 5 were computed from the representations when frozen for transfer, but one might wonder what are the dynamics of the properties over time. Recall that we froze and transferred each representation after 100 episodes were completed in the training phase. This choice balances the need for reasonable performance without having to select somewhat

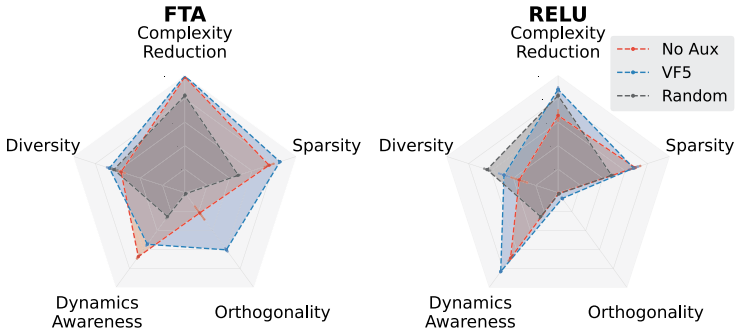


Fig. 7. The VirtualVF5 task produces bad FTA representations but improves ReLU representations. Each subplot shows a property value achieved by four different representations: FTA and ReLU with VirtualVF5, and FTA and ReLU with no auxiliary tasks. It is clear this auxiliary task changes the properties of the representations; particularly Orthogonality. We did not include non-interference as VirtualVF5 had no impact on it.

arbitrary steps budgets or performance criteria. However, our choice does mean that each representation could receive different amounts of experience, and thus begs the natural question: would the properties reported in Fig. 5 be very different with more or less training. Fig. 6 provides the answers.

Generally, across all auxiliary tasks and activation functions, the representation property values remained similar after initial transients in early learning. Each subplot of Fig. 6 shows a particular property value for every single representation tested over an extended training period. We intentionally do not distinguish between activation functions and auxiliary tasks in this plot. The change in color indicates when the representation was frozen for transfer, in terms of training time. Note, many representations were frozen after the same number of training steps. Orthogonality, dynamic awareness, and sparsity of a small number of representations slowly increases with more training and complexity reduction of a few representations slowly decreases. Note, we did not anneal the learning rates overtime, so this convergence of properties cannot be explained by an artificial cessation of learning. Overall, the properties for the most part converge, and do so just before the representations were frozen for transfer. Training the representations longer would not have resulted in significant changes to the property values.

Finally, we investigated why VirtualVF5 was helpful for ReLU and harmful for FTA through the lens of representation properties. We plot the properties of these representations in Fig. 7. Interestingly, the addition of this auxiliary loss minorly decreased dynamics awareness for FTA, but increased it for ReLU. In the previous results (Fig. 5), we have observed that representations with high dynamic awareness usually do not perform well in transfer. Additionally, VirtualVF5 caused the FTA-based representation to have much higher orthogonality. For ReLU with VirtualVF5, the orthogonality increased minorly. It is unclear why this auxiliary loss caused these changes in properties, but this consistent change in the properties helps understand this outcome.

9. Transfer across Atari 2600 game modes

In this section we investigate our methodology and properties on a larger environment, with a high-performance agent that is substantially more complicated. So far, in order to perform a thorough empirical evaluation, our analysis focused on a small, controllable pixel-based navigation environment. To demonstrate the generality of our approach, we apply our methodology to understand representations learned for transfer across different Atari 2600 game modes [7,47] (see Fig. 8). We chose this setting because transfer in Atari 2600 games is an active research question, with some recent promising results [68] when transferring policies learned by a state-of-the-art agent called Rainbow [26]. We used the same three Atari games: Freeway, Space Invaders and Breakout.

Rainbow can be seen as an auxiliary task learning agent, with auxiliary heads produced by two value heads (dueling networks [86]) and estimates of the distribution of returns (C51 [6]). The agent also learns with non-linear function approximation. It uses a convolutional network as the representation function, and fully connected layers as the value function. More agent details are given in Appendix D.

The experimental protocol is very similar to before. We still learn in one source task (the default game mode 0), where the agent learns for 200 million frames, with the learned policy evaluated every iteration for 500,000 steps. In the target task (a different game mode), the representation is frozen and the agent learns the values from scratch for 10 million frames. We contrast the representations and transfer performance of Rainbow to a simple baseline: a random representation. We take the exact same Rainbow agent, with the same network architecture, and fix the representation to its random initialization. The value function is still updated in the same way, using dueling networks, distributional updates, prioritized replay and so on.

Previous transfer results [68] investigated transferring both the learned values and the representation, and they allowed for fine-tuning. Rusu et al. [68] found Rainbow policies can transfer. We are asking a different question: can the learned representation transfer and what are its properties?

We first look at the transfer performance of Rainbow with a learned representation, and Rainbow with a fixed random representation, in comparison to learning from scratch. In Fig. 9 we see two clear outcomes. In general, we see successful transfer: the initial learning phase in the source task resulted in improved performance for Rainbow, as compared to learning from scratch (if the blue line is above the black line positive representation transfer is observed). It was not entirely obvious that this positive representation

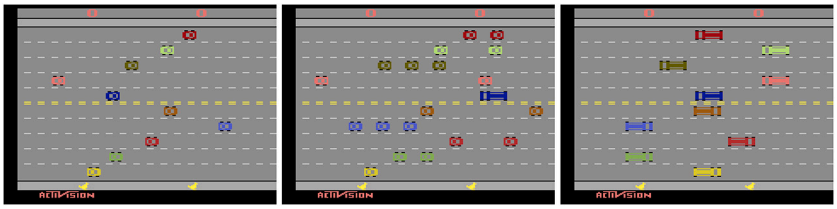


Fig. 8. Three different modes of the Atari game FREEWAY. Atari 2600 cartridges allowed players to choose between game variations that modified sprites, velocities, etc. These different modes are an excellent platform for investigating transfer. They are free from experimenter's bias and, because of hardware limitations, the different modes of a game were often small variations of the default mode, thus are not too different from one another. In this particular example, in FREEWAY, the default mode (left is m0) consists of a thin traffic in which there are only cars and these cars move at a relatively slow speed. The mode in the center (m1) has trucks, traffic is heavier and moves much faster. The mode on the right (m3) has trucks in all lanes with varying speeds.

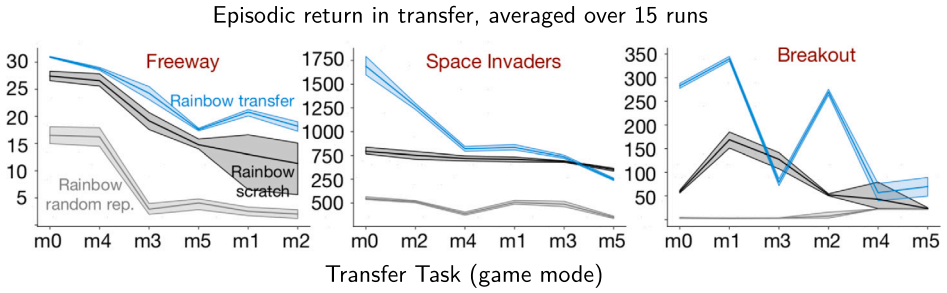


Fig. 9. Transfer in three Atari games, to game modes of increasing difficulty. The episodic return (computed over 500,000 steps) of the greedy policy is obtained every 1 million training steps, then averaged into a single performance number (AUC), plotted on the y-axis. Shaded regions correspond to standard 95% confidence intervals. The source task, default game mode zero, is plotted first. The remaining modes are ordered by the AUC achieved by training rainbow from scratch. In some cases, the default game mode appears to be more difficult, making performance not monotonically decreasing. Some of the gaps between Rainbow and Scratch look small, but they are actually substantial. They look smaller only because Random baseline does so poorly, skewing the scale of the plot. Notice that in Freeway, the representation even transferred to quite different modes (e.g., m1 and m3 visualized in Fig. 8), with heavier traffic, different vehicle speeds and vehicle types (trucks).
.

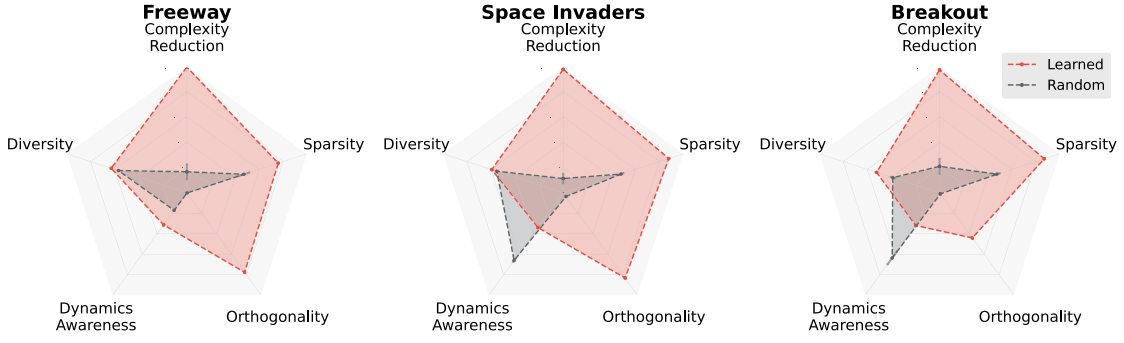


Fig. 10. Properties for the representations learned by Rainbow in Atari.

transfer would be possible, as prior work transferred the entire network and used fine-tuning. Second, the random representation generally does not allow the agent to learn in the target task, indicating learning features in the source mode facilitated transfer. This result is surprisingly similar to our results with DQN in the maze (compare Fig. 3 and Fig. 9).

Next, we consider the properties of the representation produced by Rainbow in the source game mode. We contrast these properties to the representation at initialization, to see how much Rainbow changed the representation properties after training. In the radar plots in Fig. 10, we see that in general Rainbow increased all the properties, except for dynamics awareness. The most stark change was in complexity reduction, which was significantly increased (see Table E.3 in Appendix E for unnormalized complexity–L_rep–results). This matches our previous results in the maze, where high complexity reduction was key. We also see that Rainbow significantly increases sparsity and orthogonality. Recall that one of our hypotheses for the success of FTA was the fact that it was more naturally able to obtain higher sparsity and orthogonality. We see that diversity only slightly increases, similar to our results under FTA, though different from our previous ReLU network. Both FTA and Rainbow produced more sparse and orthogonal representations than the ReLU network, which may explain why diversity increased for them both but not for ReLU.

Finally, dynamics awareness was brought to a medium-low value for all environments (increased in Freeway and significantly decreased in the other two). This outcome is different from what we observed in the maze experiments, where dynamics awareness increased. One difference is that the random representations for Space Invaders and Breakout start with relatively high dynamics

awareness: a random initialization makes nearby points look more similar and more distant points more dissimilar. Decreasing the dynamics awareness could happen either due to making temporally distant points have more similar representations or to making representations for successor states very different. We can speculate further why this happens. In the Maze and in Freeway the learned representations exhibited higher dynamics awareness compared to a random representation. In both environments the dynamics are regular throughout training: the location of walls and the movement of cars are independent of the agent's actions. Breakout and Space Invaders are very different. The games start with many objects on the screen and a successful agent learn to destroy all of them. A well trained representation might produce a smoothly generalizing function with mostly similar states along the trajectory of a well played game. Regardless of the explanation, our result suggests high dynamics awareness is environment specific and perhaps not necessary for good performance.

10. Conclusion

The goal of this work is to make progress towards an answer to a classic question: how do the properties of representations, that emerge under standard architectures used in reinforcement learning, relate to the transfer performance? We introduced a method of measuring the similarity between training and transfer tasks and designed experiments to assess learned representations. All tasks are similar, in that they involve navigating to locations in the same Maze. Intuitively, transfer should be possible, even to locations that are quite far from the goal in training. We found that 1) ReLU-based representations transferred only to very similar tasks, 2) some auxiliary tasks improved transfer of ReLU-based representations, but none facilitated transfer to less similar tasks, 3) the FTA activation significantly improved transfer, suggesting it might be a promising activation to use going forward, and 4) transfer was not possible with a linear value function, even in this seemingly simple environment.

We extensively and systematically investigated the properties of all of these (good and bad) representations attempting to better understand the correlation between representation properties and the improvement in transferability. We defined diversity, complexity reduction, and dynamics awareness, as well as used measures of orthogonality, sparsity, and non-interference from the literature. In general, interim values for properties were better: representations at the very extremes were never the best. Further, we found that the best representations maintained high complexity reduction, medium-high dynamics awareness, medium diversity, orthogonality, and sparsity. These conclusions do not mean representations should have medium orthogonality, for example, but rather representations that, in our transfer setting, emerge under training with auxiliary losses tend to do more poorly if orthogonality is higher or if it is very small (at the extreme).

One message from this work is the importance to use a simpler environment to develop a systematic methodology and obtain comprehensive results. Even in just this setting, there was a mountain of data to analyze. Further, results in this simple environment were already informative and changed our perspective on these representations. A priori, one might have thought that transfer would be very easy in this environment. Yet, repeatedly we hit roadblocks. The specific conclusions about network architectures and activations, auxiliary losses, and even properties, may be different in other environments, but the higher-level conclusions about the relevance of these properties, the interactions between components, and the need for a careful methodology to understand these nuances extends.

We demonstrated the generality of the approach by using the same methodology to analyze the representation of a Rainbow agent that transfers across Atari modes. We found that—similarly to the high-performing representations in the maze environment—the representation had high complexity reduction, higher levels of orthogonality and sparsity, and interim diversity. A key difference was that the dynamics awareness was medium-low, rather than medium-high. This difference might indicate that this property is useful for certain environments, like cost-to-goal environments, and not a useful property for representations to have in other environments, like Atari games.

There are many possible next steps to use this methodology to improve our understanding of representations in reinforcement learning. There has been substantial effort to characterize transfer, generalization, and overfitting in deep reinforcement learning, primarily in terms of performance [57,64,10]. A natural next step is to build on this work: repeat their experiments and measure the properties of the learned representations. Another important next step is to consider correlation between properties and transfer when fine-tuning the representation. In this work, we tested fixed features, to ask if initial learning produced useful features for future learning. It is also useful to ask if initial learning produced a useful starting point or initialization for future learning. We may find that different representation properties are useful in this fine-tuning regime. Moreover, investigating the representations learned in more visually complex and possibly partially observable environments and with recurrent neural networks would be also helpful.

CRediT authorship contribution statement

Han Wang: Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing, Validation. **Erfan Miahi:** Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing, Validation. **Martha White:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing, Visualization. **Marlos C. Machado:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing, Visualization. **Zaheer Abbas:** Data curation, Formal analysis, Investigation, Methodology, Software, Visualization. **Raksha Kumaraswamy:** Data curation, Methodology. **Vincent Liu:** Data curation, Methodology. **Adam White:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Han Wang reports financial support was provided by Canadian Institute for Advanced Research. Han Wang reports financial support was provided by Alberta Machine Intelligence Institute. Han Wang reports financial support was provided by Natural Sciences and Engineering Research Council of Canada.

Data availability

Data will be made available on request.

Acknowledgements

We thank Vlad Mnih for reviewing an earlier version of this paper. This work was generously funded by the CIFAR Canada AI Chair program, Amii, and NSERC. We also thank the Digital Research Alliance of Canada for the hardware used to running the maze experiments.

Appendix A. More on some representation properties

Interestingly, orthogonality in the representations, as measured in the paper, has a strong relationship with three other properties. Specifically, orthogonality in the representations can be the equivalent to orthogonality between features, which implies that each feature captures distinct information from the input states. Orthogonality can also result in interference, where interference measures how much an update on one state interferes with the updates on other states. Lastly, orthogonal features can be indicators of linearly uncorrelated features.

In this section, we show the aforementioned relationships between orthogonal representations, orthogonal features, interference, and uncorrelated features. To do so, we make the assumption of having a finite set of input-states $\{s_1, s_2, \dots, s_n\}$ represented in a d-dimensional space, where the corresponding representations are $\phi(s_i) = [f_1(s_i), f_2(s_i), \dots, f_d(s_i)]^\top$ (f_l being a function to produce the l-th feature dimension), and features are $\psi(f_i) = [f_i(s_1), f_i(s_2), \dots, f_i(s_n)]^\top$.

A.1. Relationship between orthogonal representations and orthogonal features

Here, we show that there is an equivalence relationship between orthogonality in the representation and between features. Below we show that $\sum_{i,j}^{n} (\phi(s_i)^\top \phi(s_j))^2 = \sum_{k,l}^{d} (\psi(f_k)^\top \psi(f_l))^2$.

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

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

A.2. Relationship between orthogonal representations and interference

Under linear updating, orthogonal representations would reduce interference. This is because, if two states s_1 and s_2 have $\phi(s_1)^\top \phi(s_2) = 0$, then performing an update in s_1, $\bar{w} = w + \alpha\delta\phi(s_1)$ has no impact on the prediction in s_2: $\phi(s_2)^\top \bar{w} = \phi(s_2)^\top (w + \alpha\delta\phi(s_1)) = \phi(s_2)^\top w + \alpha\delta\phi(s_2)^\top \phi(s_1) = \phi(s_2)^\top w$.

A.3. Relationship between orthogonal and uncorrelated features

Here, we show the relationship between orthogonal features and uncorrelated features. Let $\bar{\psi}(f_i) = [\bar{f_i}, \bar{f_i}, \ldots, \bar{f_i}]^\top$, where $\bar{f_i} = \frac{1}{n}\sum_{j=1}^{n}[f_i(s_j)]$, denote the expected value of feature i over the set of input-states. If all features are centered, that is, $\bar{f_i} = 0$ for all i, then it is trivial to see that

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

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

Appendix B. Empirical details

B.1. Auxiliary tasks

In this section, we provide a more detailed explanation of each of the seven auxiliary tasks used for helping with representation learning. All the losses use the same samples taken from the replay buffer to update the value function, except the ATC loss. The detailed formulas for the auxiliary losses are presented in Table B.1.

Augmented Temporal Contrast (ATC) Loss: This contrastive loss encourages the network to learn similar representations for an input state, s_t, with one from a pre-determined, near-future time step input state, s_{t+k}, where $k = 3$. In contrast to other losses that we have employed in this work, this loss uses more than a single auxiliary head to compute itself. To do so, it first applies a data augmentation technique called random shift with a probability of 0.1 and padding of 4 to both of these input states. Then, it feeds the augmented version of s_t, AUG(s_t), through a set of networks to compute p_t, where F_{θ_A} is a linear mapping of representation into an embedding space with a size of 32, and F_{θ_C} is a single layer neural network with a hidden layer size of 64, and an output size of 32. Then, AUG(s_{t+k}) is fed into a momentum encoder of Φ_{θ_R} ($\phi_{\breve{\theta}_R}$) and F_{θ_A} ($F_{\breve{\theta}_A}$) to compute c_{t+k}. The output of these networks, p_t and c_{t+k}, are combined with each other through a 32×32 matrix called W to compute logits, $l_{i,j+k}$. At last, these logits are used to compute the InfoNCE loss, $\mathcal{L}_{ATC}$. In contrast to the original paper that uses a complex learning-rate scheduling technique, we implement this loss in its simplest form by using a fixed learning-rate. We sweep through values of $[0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001]$ of this learning-rate. We update the momentum encoder in every step using a τ of 0.01. The batch size is the same as the batch size used for computing the value function loss, and the weight of this auxiliary loss is set to 1.

Input Reconstruction (IR): This auxiliary task reconstructs the input image from the representation through a deconvolutional network. On the auxiliary head, the representation was firstly projected to a hidden layer with 1024 nodes, then sent to a two layer deconvolutional neural network, with 4 kernels, 32 and 3 channels, 2 and 1 strides, 2 and 1 pads on 2 layers separately. The output had the same size as the input image (i.e., $15 \times 15 \times 3$). The weight of the auxiliary loss was set to 0.0001.

Next Agent State Prediction (NAS): This task motivates the representation of the current state to minimize its prediction error on the next state representation. To do so, it uses a contrastive loss that minimizes the prediction error on the next state while maximizes the prediction error on the rest of the states. For this task, the auxiliary head, F_{θ_A}, consists of two fully connected layers with 64 neurons on each. The weight of this auxiliary loss was 0.001.

Successor Feature Prediction (SF): Successor feature prediction task was similar to next-agent-state prediction, though the target was constructed by bootstrapping. Given the transition on latent space $< \phi_t, a_t, \phi_{t+1}, a_{t+1} >$, the auxiliary head learned to minimize the difference between the prediction $F_{\theta_A}(\phi_t, a_t)$ and the target $(1-\lambda)\phi_{t+1} + \lambda F_{\theta_A}(\phi_{t+1}, a_{t+1})$, where λ is set to 0.99. To satisfy the property of successor features, we added an extra head to predict the reward linearly from the representation ϕ_t. This head for predicting the successor feature used the same neural network architecture as NAS. The weight was set to 1.

Reward Prediction (Reward): Another auxiliary prediction task was to predict the reward independently given the representation ϕ_t. This auxiliary task used the same non-linear transformation structure as SF and the weight is set to 1.

Expert Target Prediction (XY): The last prediction task was expert-designed targets prediction. The agent was asked to predict its current position given the image. Since predicting the position was considered a regression task, we used MSE loss with the same network structure as Reward. We set a low weight of 0.0001 for this auxiliary task.

Virtual Value Function Learning (VirtualVF): This auxiliary task learns a different value function on the auxiliary head. There were 2 settings in Maze—learning a value function assuming the goal is on grid $[7, 7]$, and learning 5 value functions when the goals are on grid $[0, 0]$, $[0, 14]$, $[14, 0]$, $[14, 14]$, $[7, 7]$ separately. The weight of this auxiliary task remained 1 but the discount rate on the auxiliary head was set to be lower, 0.9, so that the agent can focus on the main task. The auxiliary head learned this task with the same network structure as XY.

Table B.1
Auxiliary losses used in this paper, defined with respect to transitions in D.

ATC	$\mathcal{L}_{ATC}(D) = -\mathbb{E}_S\left[\log \frac{\exp l_{i,i+k}}{\sum_{j \in S} \exp l_{i,j+k}}\right]$, where $l_{t,k}$ are logits computed bilinearly such that $l_{i,j+k} = p_i W c_{j+k}$ and $p_t = F_{\theta_C}(F_{\theta_A}(\Phi_{\theta_R}(\text{AUG}(s_t)))) + F_{\theta_A}(\Phi_{\theta_R}(\text{AUG}(s_t)))$, $c_{t+k} = F_{\theta_A}(\phi_{\theta_R}(\text{AUG}(s_{t+k})))$
IR	$\mathcal{L}_{\text{IR}}(D) = \mathop{\mathbb{E}}_{s \sim D}\left[\|F_{\theta_A}(\Phi_{\theta_R}(s)) - s\|_2^2\right]$
NAS	$\mathcal{L}_{\text{NAS}}(D) = \mathop{\mathbb{E}}_{\substack{(s_t,a_t,s_{t+1}) \sim D \\ (s_k,a_k,s_{k+1}) \sim D}}\left[\|F_{\theta_A}(\Phi_{\theta_R}(s_t), a_t) - (\Phi_{\theta_R}(s_{t+1}) - \Phi_{\theta_R}(s_t))\|_2^2 + \max\left(0, 1 - \|F_{\theta_A}(\Phi_{\theta_R}(s_k), s_k) - (\Phi_{\theta_R}(s_{t+1}) - \Phi_{\theta_R}(s_t))\|_2^2\right)\right]$
Reward	$\mathcal{L}_{\text{Reward}}(D) = \mathop{\mathbb{E}}_{(s_t,a_t,r_{t+1}) \sim D}\left[\|F_{\theta_A}(\Phi_{\theta_R}(s_t), a_t) - r_{t+1}\|_2^2\right]$
SF	$\mathcal{L}_{\text{SF}}(D) = \mathop{\mathbb{E}}_{(s_t,a_t,s_{t+1},a_{t+1}) \sim D}\left[\|F_{\theta_A}(\Phi_{\theta_R}(s_t), a_t) - (\Phi_{\theta_R}(s_t) + \gamma F_{\hat{\theta}_A}(\Phi_{\hat{\theta}_R}(s_{t+1}), a_{t+1}))\|_2^2\right]$
VirtualVF	$\mathcal{L}_{\text{VirtualVF}}(D) = \sum_{g \in \mathcal{G}} \mathop{\mathbb{E}}_{(s_t,a_t,r_{t+1},s_{t+1},a_{t+1}) \sim D}\left[\|F^g_{\theta_A}(\Phi_{\theta_R}(s_t), a_t) - (r^g_t + \gamma F^g_{\hat{\theta}_A}(\Phi_{\hat{\theta}_R}(s_{t+1}), a_{t+1})\|_2^2\right]$
XY	$\mathcal{L}_{\text{XY}}(D) = \mathop{\mathbb{E}}_{(s_t,x_t,y_t) \sim D}\left[\|F_{\theta_A}(\Phi_{\theta_R}(s_t)) - (x_t, y_t)\|_2^2\right]$
	$\mathcal{G}$ is a set of goal locations. The reward r_g and representation function $\phi^g_{\theta_A}$ are associated with the goal g, sampled from this set.

Table B.2
Representation size settings.

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

B.2. General hyperparameter setting

We used the same neural network architecture across representations learned with the various loss functions, for each domain. All hidden layers are initialized with Xavier. Table B.2 shows the number of nodes on the representation function's last hidden layer, and the number of features.

During training, the inputs are normalized to be in the range $[-1, 1]$. We use Adam optimizer to update weights, and we used the mean-squared error as the loss. The batch size is set to be 32. The buffer has length $10,000$. The input image is normalized. For the representation function, we use a two layer convolutional network with kernel size of 4, stride of 1, padding of 1, and 32 channels for the first layer; kernel size of 4, stride of 2, padding of 2, and 16 channels on the second layer. A target network was used with the synchronization frequency set to 64. The buffer's memory size was $100,000$ and there were 32 samples randomly chosen at each step. The agent learns for $300,000$ steps with ϵ-greedy policy. During transfer learning, all agents, including baselines, learned for $100,000$ steps only.

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

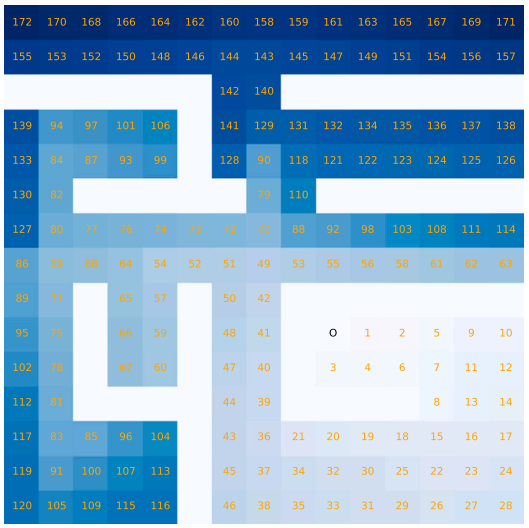


Fig. B.11. Transfer tasks are ranked by the similarity between successor features of each task. This figure shows the similarity ranking of different tasks compared to the source task, where the source task is marked by O. Each number in the cell indicates the similarity rank of the task when the goal is moved to that specific position.

The learning rate was swept for every representation learning architecture and control task. The best setting was picked according to the averaged performance over 5 runs. Each run uses a different random seed. We tested the learning rate in $[0.001, 0.0003, 0.0001, 0.00003, 0.00001]$ during the representations training step and for learning from scratch baseline agents. During transfer, we tested $[0.01, 0.003, 0.001, 0.0003, 0.0001]$ for representations with 32 features and $[0.001, 0.0003, 0.0001, 0.00003, 0.00001]$ for all representations with 640 features, as there were more weights to train, which required more careful learning.

In the non-linear value function case and transfer tasks with linear value function, we use a fixed $\epsilon = 0.1$. However, in representation learning in the linear case, it turned out to be harder for the agent to converge when keeping other settings as the same as in the non-linear value function. Thus, we provided a better exploration in the early learning stage to speed up learning by decreasing ϵ, which decreases from 1 to 0.1 in the first $100,000$ steps.

B.3. Using SFs to measure task similarity

When checking how the difficulty of transfer affects the transfer performance, we consider the similarity between each transfer task to the original task. When a transfer task is similar to the task in which the representation is trained, we consider the transfer to be easier.

We measure the similarity between tasks according to the successor representations, ψ. Since successor representations encode the trajectories of the agent, the difference between successor representations generated by optimal policy can reflect the difference between optimal policies that the agent learns in different tasks. If the optimal policies of two tasks turn out to be dissimilar, the similarity between these tasks is considered low.

To compute a highly accurate estimate of successor features, we solve the maze by using a simple tabular algorithm. To do so, we define the state as the cell in the maze that is occupied by the agent. Doing so results in having 173 states in total. Taking this into account, we use value iteration to generate an optimal policy, then calculate the successor representation of each state based on the optimal policy.

The successor representations of all states in the same task are considered. The successor representation of each task, Ψ, is obtained by concatenating all successor features in the same task. The similarity is defined as the dot product between Ψ's in the transfer and the original tasks. We choose the dot product to keep both angle and magnitude information between concatenated successor representations. A higher dot product value means the transfer task is more similar to the original task and vice versa.

$$\psi(s, task_x) = \mathbb{E}_{\pi^*_{task_x}}\left[\sum_{t=0}^{T} \gamma^t f(S_t) | S_0 = s\right]$$

$$\Psi_{task_x} = \left[\psi(s_0)\, \psi(s_1)\, \cdots\, \psi(s_{|S|})\right]$$

$$similarity(task_x, task_y) = \Psi_{task_x} \cdot \Psi_{task_y}$$

Interestingly, the goal states that are more distant from each other become more dissimilar by computing the similarity this way. This is more clear when we take a look at the similarity rankings of the goal states, as depicted in Fig. B.11. As shown in Fig. 3, the representations have a hard time transferring to higher-ranking goals, so there is a clear connection between the ranks of the transfer tasks and the transfer performance. These findings support the use of this approach for calculating task similarity and ranking.

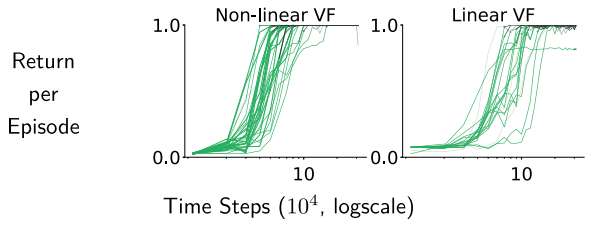


Fig. B.12. Return converges in the representation training step. The plot shows the averaged return over the most recent 100 episodes at each checkpoint. The x-axis is the number of time steps and the y-axis is the average return. Each curve represents one agent specification (activation and auxiliary task pair). As our main focus is not to compare the learning efficiency during the representation learning step, and the difference between learning curves is not large, we only show the general trend by plotting every curve with the same color. The curve changes color to black, at the time point where we took the representation and fixed it.

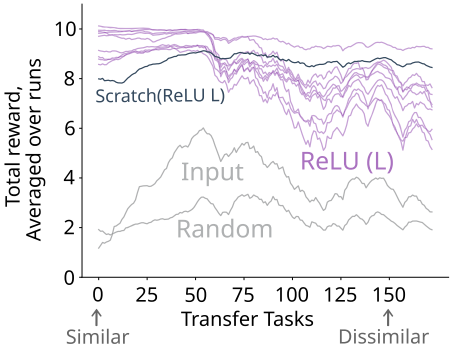


Fig. C.13. Transfer performance of larger ReLU representations (45 in total) on 173 transfer tasks. The tasks on the x-axis are arranged by similarity to training tasks: on the left (small x-values) being most similar and on the right (large x-values) being most dissimilar. The black line shows the performance when learning in each transfer task from scratch, with the same representation size as ReLU(L). Lines completely above the black line indicate a representation yielded successful transfer in all tasks. Lines that start above the black line but fall below it as we move left to right indicate a representation that transfers to similar tasks but not dissimilar tasks.

Appendix C. Additional results

C.1. Representation training

We show the learning curve of all representation learning architectures in Fig. B.12, to show that the early-saved representations have converged when they are saved.

C.2. Larger ReLU transfer

We show the transfer performance of ReLU(L) in Fig. C.13. The ReLU(L) setting stays between ReLU and FTA representations: ReLU(L) keeps the same activation function as ReLU representation, but increases the size of the representation layer to 640, which is the same as the size of FTA representations. Therefore, it maintains the same value function capacity as the FTA representations. The pattern in the transfer performance of ReLU(L) is similar to ReLU (Fig. 3). As the transfer tasks become dissimilar, the transfer performance drops below the Scratch agent. In general, when considering the total reward obtained by the agent, the performance of ReLU(L) is better than ReLU and worse than FTA.

C.3. Transfer with different architectures

Fig. C.14 shows the 95% confidence interval of transfer performance with different representation sizes, activation functions, and different auxiliary tasks, with a non-linear value function.

C.4. Relationship between properties

We also checked the relationship between properties. The result is shown in Fig. C.15. Two subplots are highlighted with the orange color. We noticed diversity and complexity reduction showed strong positive linear correlation. This suggests that monitoring either diversity or complexity reduction should be informative to predict the dissimilar task transfer performance in practice. Furthermore, a threshold exists when looking at diversity and orthogonality, as well as diversity and complexity reduction. For representations with higher diversity (higher than 0.5, in this case), it also showed higher orthogonality, while this pattern does not exist in low diversity representations. Although there exist several outliers, this still indicates the possibility that pursuing a representation with high orthogonality may result in a relatively high diversity at the same time in practice.

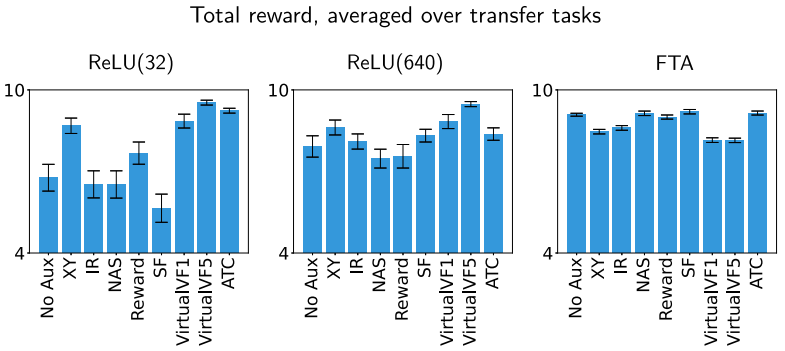


Fig. C.14. Transfer performance depends on the activation function, representation size, and auxiliary tasks. This plot presents the same data as Fig. 4, but the error bar shows a 95% confidence interval. The bar shows the mean value over 5 seeds × 173 transfer tasks.

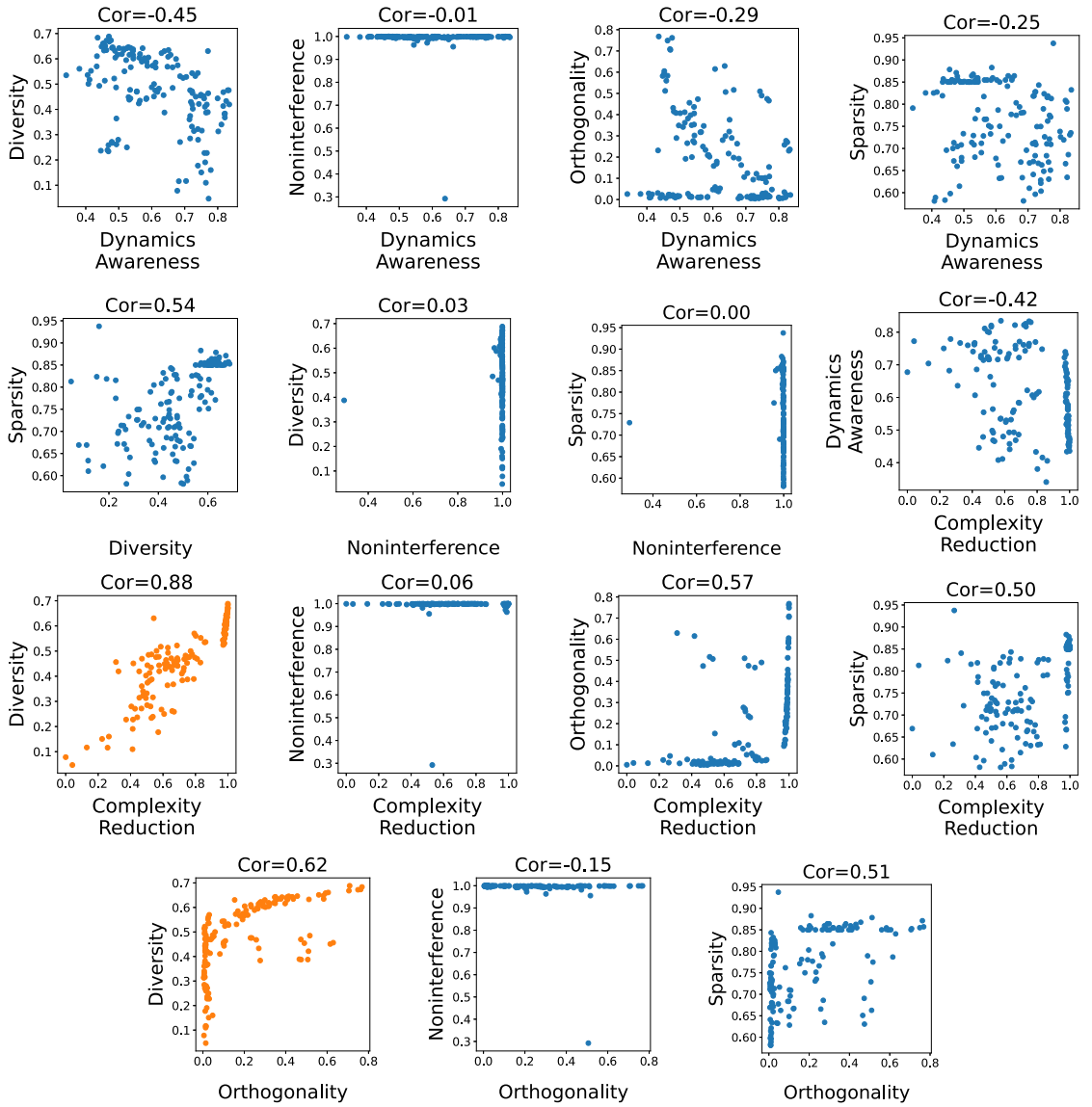


Fig. C.15. There exists strong positive correlation between diversity and complexity reduction. Meanwhile, when diversity is high, the increment of diversity comes with an increment of orthogonality. The above 2 subplots are highlighted with orange color. Each subplot shows the relationship between a pair of properties. Each scatter refers to one representation. The x and y coordinates are property measures. The properties' names are shown as labels on each axis. The correlations (Cor) are reported above each subplot.

Appendix D. Experimental details for Atari

We used Rainbow-DQN with the default hyper-parameter setting provided by Hessel et al. [26]. The rainbow agent combines DQN [51] double Q-learning [25], prioritized experience replay with the proportional variant [71], dueling networks [86], noisy networks for exploration [20], and distributional RL [6]. The agent is similar to the Rainbow-IQN agent used in the recent work showing transfer across modes [68] except that it uses C51 for distributional RL [6] instead of IQN [11].

We interpret the convolutional layers of the network of the Rainbow agent as the representation network and the successive fully connecting layers as the value network. The architectural details are worth mentioning here. The representation network consists of three convolution layers each followed by ReLU activations, and produces a representation of 3136 features, as a function of stacked input frames (see Hessel et al. [26] for the number of kernels and stride configuration). The value-network consists of separate fully-connected networks that estimate value and advantage [86]. The value network takes as input the output of the representation network and the outputs of the value network are combined to estimate the value distribution [6]. The fully-connected networks that estimate value and advantage have a single hidden layer of 512 units, with noisy linear networks [20] and ReLU activations.

Appendix E. Additional complexity results in Atari

We put the raw complexity in Atari in Table E.3.

Table E.3
Unnormalized complexity (L_rep) of the learned and random Rainbow representations in Freeway, Space Invaders, and Breakout (average of 5 representations).

Game	Learned Representation	Random Representation
Freeway	0.029	4.178
Space Invaders	0.228	6.908
Breakout	0.229	6.034

References

[1] R. Agarwal, M.C. Machado, P.S. Castro, M.G. Bellemare, Contrastive behavioral similarity embeddings for generalization in reinforcement learning, in: International Conference on Learning Representations, 2021.
[2] D.R. Ashley, S. Ghiassian, R.S. Sutton, Does standard backpropagation forget less catastrophically than Adam?, arXiv preprint, arXiv:2102.07686, 2021.
[3] A. Atrey, K. Clary, D. Jensen, Exploratory not explanatory: counterfactual analysis of saliency maps for deep reinforcement learning, in: International Conference on Learning Representations, 2020.
[4] A. Barreto, W. Dabney, R. Munos, J.J. Hunt, T. Schaul, H.P. van Hasselt, D. Silver, Successor features for transfer in reinforcement learning, in: Advances in Neural Information Processing Systems, 2017.
[5] M. Bellemare, W. Dabney, R. Dadashi, A.A. Taiga, P.S. Castro, N. Le Roux, D. Schuurmans, T. Lattimore, C. Lyle, A geometric perspective on optimal representations for reinforcement learning, in: Advances in Neural Information Processing Systems, 2019.
[6] M.G. Bellemare, W. Dabney, R. Munos, A distributional perspective on reinforcement learning, in: International Conference on Machine Learning, 2017.
[7] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, J. Artif. Intell. Res. 47 (2013) 253–279.
[8] R. Caruana, Multitask learning, in: Machine Learning, 1997.
[9] W. Chung, S. Nath, A. Joseph, M. White, Two-timescale networks for nonlinear value function approximation, in: International Conference on Learning Representations, 2019.
[10] K. Cobbe, O. Klimov, C. Hesse, T. Kim, J. Schulman, Quantifying generalization in reinforcement learning, in: International Conference on Machine Learning, 2019.
[11] W. Dabney, M. Rowland, M. Bellemare, R. Munos, Distributional reinforcement learning with quantile regression, in: AAAI Conference on Artificial Intelligence, 2018.
[12] T. Dai, K. Arulkumaran, S. Tukra, F. Behbahani, A.A. Bharath, Analysing deep reinforcement learning agents trained with domain randomisation, arXiv preprint, arXiv:1912.08324, 2019.
[13] P. Dayan, Improving generalization for temporal difference learning: the successor representation, Neural Comput. (1993).
[14] S.S. Du, S.M. Kakade, R. Wang, L.F. Yang, Is a good representation sufficient for sample efficient reinforcement learning?, in: International Conference on Learning Representations, 2019.
[15] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, in: Advances in Neural Information Processing Systems, 1990.
[16] A.m. Farahmand, A. Barreto, D. Nikovski, Value-aware loss function for model-based reinforcement learning, in: Artificial Intelligence and Statistics, 2017.
[17] J. Farebrother, M.C. Machado, M. Bowling, Generalization and regularization in DQN, arXiv preprint, arXiv:1810.00123, 2018.
[18] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, 2017.
[19] C. Finn, X.Y.T. Tan, Y. Duan, T. Darrell, S. Levine, P. Abbeel, Deep spatial autoencoders for visuomotor learning, in: IEEE International Conference on Robotics and Automation, 2016.
[20] M. Fortunato, M.G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, S. Legg, Noisy networks for exploration, in: International Conference on Learning Representations, 2018.
[21] V. François-Lavet, Y. Bengio, D. Precup, J. Pineau, Combined reinforcement learning via abstract representations, in: AAAI Conference on Artificial Intelligence, 2019.
[22] R.M. French, Catastrophic forgetting in connectionist networks, Trends Cogn. Sci. (1999).
[23] S. Greydanus, A. Koul, J. Dodge, A. Fern, Visualizing and understanding Atari agents, in: International Conference on Machine Learning, 2018.
[24] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, J. Malik, Cognitive mapping and planning for visual navigation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017.
[25] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: AAAI Conference on Artificial Intelligence, 2016.
[26] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M.G. Azar, D. Silver, Rainbow: combining improvements in deep reinforcement learning, in: AAAI Conference on Artificial Intelligence, 2018.

[27] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, A. Lerchner, Towards a definition of disentangled representations, arXiv preprint, arXiv: 1812.02230, 2018.
[28] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, A. Lerchner, DARLA: improving zero-shot transfer in reinforcement learning, in: International Conference on Machine Learning, 2017.
[29] G.Z. Holland, E.J. Talvitie, M. Bowling, The effect of planning shape on Dyna-style planning in high-dimensional state spaces, arXiv preprint, arXiv:1806.01825, 2018.
[30] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, Reinforcement learning with unsupervised auxiliary tasks, in: International Conference on Learning Representations, 2017.
[31] K. Javed, M. White, Meta-learning representations for continual learning, in: Advances in Neural Information Processing Systems, 2019.
[32] P. Kanerva, Sparse Distributed Memory, MIT Press, 1988.
[33] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: International Conference on Learning Representations, 2015.
[34] V.R. Kompella, M. Luciw, J. Schmidhuber, Incremental slow feature analysis: adaptive low-complexity slow feature updating from high-dimensional input streams, Neural Comput. (2012).
[35] G. Konidaris, S. Osentoski, P. Thomas, Value function approximation in reinforcement learning using the Fourier basis, in: AAAI Conference on Artificial Intelligence, 2011.
[36] T. Kurutach, A. Tamar, G. Yang, S.J. Russell, P. Abbeel, Learning plannable representations with Causal InfoGAN, in: Advances in Neural Information Processing Systems, 2018.
[37] T. Lattimore, C. Szepesvari, G. Weisz, Learning with good feature representations in bandits and in RL with a generative model, in: International Conference on Machine Learning, 2020.
[38] E. Lecarpentier, D. Abel, K. Asadi, Y. Jinnai, E. Rachelson, M.L. Littman, Lipschitz lifelong reinforcement learning, in: AAAI Conference on Artificial Intelligence, 2021.
[39] T. Lesort, N. Díaz-Rodríguez, J.F. Goudou, D. Filliat, State representation learning for control: an overview, Neural Netw. 108 (2018) 379–392.
[40] Y. Liang, M.C. Machado, E. Talvitie, M.H. Bowling, State of the art control of Atari games using shallow reinforcement learning, in: International Conference on Autonomous Agents & Multiagent Systems, 2016.
[41] L.J. Lin, Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching, Machine Learning, 1992.
[42] V. Liu, R. Kumaraswamy, L. Le, M. White, The utility of sparse representations for control in reinforcement learning, in: AAAI Conference on Artificial Intelligence, 2019.
[43] V. Liu, A. White, H. Yao, M. White, Towards a practical measure of interference for reinforcement learning, arXiv preprint, arXiv:2007.03807, 2020.
[44] V. Liu, A.M. White, H. Yao, M. White, Measuring and mitigating interference in reinforcement learning, https://openreview.net/forum?id=26WnoE4hjS, 2021.
[45] B. Lütjens, M. Everett, J.P. How, Certified adversarial robustness for deep reinforcement learning, in: Conference on Robot Learning, 2019.
[46] M.C. Machado, M.G. Bellemare, M. Bowling, Count-based exploration with the successor representation, in: AAAI Conference on Artificial Intelligence, 2020.
[47] M.C. Machado, M.G. Bellemare, E. Talvitie, J. Veness, M.J. Hausknecht, M. Bowling, Revisiting the arcade learning environment: evaluation protocols and open problems for general agents, J. Artif. Intell. Res. 61 (2018) 523–562.
[48] M.C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, M. Campbell, Eigenoption discovery through the deep successor representation, in: International Conference on Learning Representations, 2018.
[49] S. Mahadevan, M. Maggioni, Proto-value functions: a Laplacian framework for learning representation and control in Markov decision processes, J. Mach. Learn. Res. (2007).
[50] J.L. McClelland, D.E. Rumelhart, P.R. Group, et al., Parallel Distributed Processing, Explorations in the Microstructure of Cognition: Psychological and Biological Models, vol. 2, MIT Press, 1987.
[51] 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 518 (2015) 529–533.
[52] A.V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, S. Levine, Visual reinforcement learning with imagined goals, in: Advances in Neural Information Processing Systems, 2018.
[53] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: International Conference on Machine Learning, 2010.
[54] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, J. Yosinski, Plug & play generative networks: conditional iterative generation of images in latent space, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017.
[55] J. Oh, X. Guo, H. Lee, R.L. Lewis, S. Singh, Action-conditional video prediction using deep networks in Atari games, in: Advances in Neural Information Processing Systems, 2015.
[56] J. Oh, S. Singh, H. Lee, Value prediction network, in: Advances in Neural Information Processing Systems, 2017.
[57] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, D. Song, Assessing generalization in deep reinforcement learning, arXiv preprint, arXiv:1810.12282, 2018.
[58] C. Painter-Wakefield, R. Parr, Greedy algorithms for sparse reinforcement learning, in: International Conference on Machine Learning, 2012.
[59] Y. Pan, K. Banman, M. White, Fuzzy tiling activations: a simple approach to learning sparse representations online, in: International Conference on Learning Representations, 2021.
[60] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, M.L. Littman, An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning, in: International Conference on Machine Learning, 2008.
[61] R. Parr, C. Painter-Wakefield, L. Li, M. Littman, Analyzing feature generation for value-function approximation, in: International Conference on Machine Learning, 2007.
[62] D. Pathak, P. Agrawal, A.A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: International Conference on Machine Learning, 2017.
[63] X.B. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-real transfer of robotic control with dynamics randomization, in: IEEE International Conference on Robotics and Automation, 2018.
[64] A. Rajeswaran, K. Lowrey, E.V. Todorov, S.M. Kakade, Towards generalization and simplicity in continuous control, in: Advances in Neural Information Processing Systems, 2017.
[65] B. Ratitch, D. Precup, Sparse distributed memories for on-line value-based reinforcement learning, in: European Conference on Machine Learning, 2004.
[66] C. Rupprecht, C. Ibrahim, C.J. Pal, Finding and visualizing weaknesses of deep reinforcement learning agents, in: International Conference on Learning Representations, 2020.
[67] D. Russo, B. Van Roy, Eluder dimension and the sample complexity of optimistic exploration, in: Advances in Neural Information Processing Systems, 2013.
[68] A.A. Rusu, S. Flennerhag, D. Rao, R. Pascanu, R. Hadsell, Probing transfer in deep reinforcement learning without task engineering, in: Conference on Lifelong Learning Agents, 2022.
[69] A.A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, R. Hadsell, Sim-to-real robot learning from pixels with progressive nets, in: Conference on Robot Learning, 2017.
[70] T. Schaul, D. Borsa, J. Modayil, R. Pascanu, Ray interference: a source of plateaus in deep reinforcement learning, arXiv preprint, arXiv:1904.11455, 2019.
[71] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, in: International Conference on Learning Representations, 2016.
[72] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al., Mastering Atari, Go, chess and shogi by planning with a learned model, Nature 588 (2020) 604–609.

[73] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, T. Degris, The predictron: end-to-end learning and planning, in: International Conference on Machine Learning, 2017.
[74] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, C. Finn, Universal Planning Networks: Learning Generalizable Representations for Visuomotor Control, International Conference on Machine Learning, 2018.
[75] K.L. Stachenfeld, M. Botvinick, S.J. Gershman, Design principles of the hippocampal cognitive map, in: Advances in Neural Information Processing Systems, 2014.
[76] A. Stooke, K. Lee, P. Abbeel, M. Laskin, Decoupling representation learning from reinforcement learning, in: International Conference on Machine Learning, 2021.
[77] F.P. Such, V. Madhavan, R. Liu, R. Wang, P.S. Castro, Y. Li, J. Zhi, L. Schubert, M.G. Bellemare, J. Clune, J. Lehman, An Atari model zoo for analyzing, visualizing, and comparing deep reinforcement learning agents, in: International Joint Conference on Artificial Intelligence, 2019.
[78] R.S. Sutton, Generalization in reinforcement learning: successful examples using sparse coarse coding, in: Advances in Neural Information Processing Systems, 1996.
[79] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[80] C. Szepesvári, Algorithms for Reinforcement Learning, Morgan and Claypool, 2010.
[81] E. Talvitie, Self-correcting models for model-based reinforcement learning, in: AAAI Conference on Artificial Intelligence, 2017.
[82] Y. Tang, Z.D. Guo, P.H. Richemond, B.A. Pires, Y. Chandak, R. Munos, M. Rowland, M.G. Azar, C. Le Lan, C. Lyle, et al., Understanding self-predictive learning for reinforcement learning, in: International Conference on Machine Learning, PMLR, 2023, pp. 33632–33656.
[83] S. Thrun, Is learning the n-th thing any easier than learning the first? in: Advances in Neural Information Processing Systems, 1996.
[84] J. Tyo, Z. Lipton, How transferable are the representations learned by deep Q agents? arXiv preprint, arXiv:2002.10021, 2020.
[85] Y. Wan, M. Zaheer, A. White, M. White, R.S. Sutton, Planning with expectation models, in: International Joint Conference on Artificial Intelligence, 2019.
[86] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning, 2016.
[87] M. Watter, J. Springenberg, J. Boedecker, M. Riedmiller, Embed to control: a locally linear latent dynamics model for control from raw images, in: Advances in Neural Information Processing Systems, 2015.
[88] M. White, Unifying task specification in reinforcement learning, in: International Conference on Machine Learning, 2017.
[89] Y. Wu, G. Tucker, O. Nachum, The Laplacian in RL: learning representations with efficient approximations, in: International Conference on Learning Representations, 2019.
[90] G. Yang, A. Zhang, A. Morcos, J. Pineau, P. Abbeel, R. Calandra, Plan2Vec: unsupervised representation learning by latent plans, in: Conference on Learning for Dynamics and Control, 2020.
[91] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, H. Lipson, Understanding neural networks through deep visualization, in: Deep Learning Workshop, 2015.
[92] T. Zahavy, N. Ben-Zrihem, S. Mannor, Graying the black box: understanding DQNs, in: International Conference on Machine Learning, 2016.
[93] A. Zhang, N. Ballas, J. Pineau, A dissection of overfitting and generalization in continuous reinforcement learning, arXiv preprint, arXiv:1806.07937, 2018.