DISENTANGLED (UN)CONTROLLABLE FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

In the context of MDPs with high-dimensional states, reinforcement learning can achieve better results when using a compressed, low-dimensional representation of the original input space. A variety of learning objectives have therefore been used to learn useful representations. However, these representations usually lack interpretability of the different features. We propose a representation learning algorithm that is able to disentangle latent features into a controllable and an uncontrollable part. The resulting representations are easily interpretable and can be used for learning and planning efficiently by leveraging the specific properties of the two parts. To highlight the benefits of the approach, the disentangling properties of the algorithm are illustrated in three different environments.

1 INTRODUCTION

Learning from high-dimensional data remains a challenging task. Particularly for reinforcement learning (RL), the complexity and high dimensionality of the Markov Decision Process (MDP) state often leads to complex or intractable solutions. A direct application of RL on high-dimensional input spaces therefore typically yields instabilities and poor performance. In order to still facilitate learning from high-dimensional input data, an encoder architecture can be used to compress the inputs into a lower-dimensional latent representation. To this extent, a plethora of work has successfully focused on discovering low-dimensional encoded representations that accommodate the underlying features for the task at hand (Jonschkowski & Brock, 2015; Jaderberg et al., 2017; Francois-Lavet et al., 2019; Yar, 2021; Lee et al., 2020; Schwarzer et al., 2021; Kostrikov et al., 2021; Laskin et al., 2020b;a).

The resulting low-dimensional representations however tend to seldom contain specific disentangled features, which leads to disorganized latent information. This means that the latent states can represent the information from the state in any arbitrary way, leading to non-optimal interpretability. In line with structuring a latent representation, Francois-Lavet et al. (2019) have shown notions and use of interpretability in MDP representations. When expanding this notion of interpretability to be compatible with RL, it has been argued that the agent's state should be an important element of a latent representation, since it generally represents what is controllable by the policy. In this light, Thomas et al. (2017) have introduced the concept on isolating and disentangling controllable features in a low-dimensional maze environment, by means of a selectivity loss. Furthermore, Kipf et al. (2020) took an object-centric approach to isolate distinct controllable objects. Controllable features however only represent a fragment of an environment, where in many cases the uncontrollable features are of equal importance. For example, in the context of a distribution of mazes, for the prediction of the next controllable (agent) state following an action, the information about the wall structure is crucial (See Fig. 1). A representation would therefore benefit from incorporating controllable ánd uncontrollable features, preferably in a disentangled, interpretable arrangement.

In an MDP setting, we show that a latent representation can be disentangled into two parts, where one part is designed to contain controllable features and the other part is designed to contain uncontrollable features. This allows for a precise and visible separation of the latent features, improving interpretability, representation quality and possibly moving towards a basis for building causal relationships between an agent and its environment. The learning algorithm consists of both an action-conditioned and a state-only forward predictor, along with an entropy and an adversarial loss, which reliably isolate and disentangle the controllable versus the non-controllable features. Furthermore, we show that learning and planning can achieve strong performance when it is applied on the human-interpretable disentangled latent representation.

(a) Random observations in each of the 4 mazes (b) Encoded representations $z : \ s \in \mathcal{S}$
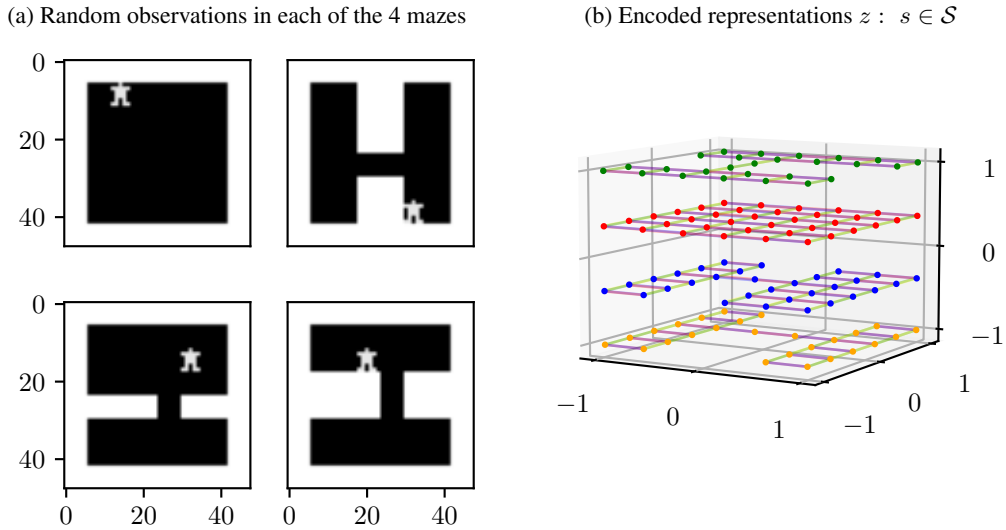

Figure 1: Visualization in a maze environment of (a) four random pixel states and (b) the disentanglement of the controllable latent $z^c \in \mathbb{R}^2$ on the horizontal axes, and the uncontrollable latent $z^u \in \mathbb{R}^1$ on the vertical axis. The representation is trained on high-dimensional pixel tuples (s_t, a_t, r_t, s_{t+1}), sampled from a replay buffer $\mathcal{B}$, gathered from random actions taken in the four maze environments shown in (a). All possible states are encoded ($z_t = f(s_t; \theta_{enc})$ and plotted in (b) with the transition prediction for each possible action, revealing a clear disentanglement between the controllable agent's position and the uncontrollable wall architecture.

2 PRELIMINARIES

We consider an agent acting within an environment, where the environment is modeled as a discrete Markov Decision Process (MDP) defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the environment's transition function, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the environment's reward mapping and γ is the discount factor. We consider the setting where we have access to a replay buffer (B) of visited states $s_t \in \mathcal{S}$ that were followed by actions $a_t \in \mathcal{A}$ and resulted in the rewards $r_t \in \mathcal{R}$ and the next states s_{t+1}. One entry in B contains a tuple of past experiences (s_t, a_t, r_t, s_{t+1}). The agent's goal is to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expectation of the discounted return $V^\pi(s) = \mathbb{E}_\tau[\sum_{t=0}^T \gamma^t R(s_t, a_t) \mid s_t = s]$, where τ is a trajectory following the policy π.

Furthermore, we examine the setting where a high-dimensional state ($s_t \in \mathbb{R}^v$) is compressed into a lower-dimensional latent state $z_t \in \mathcal{Z} = \mathbb{R}^w$ where $\mathcal{Z}$ represents the latent space with w typically smaller than v. This is done by means of a neural network encoding $f : \mathcal{S} \rightarrow \mathcal{Z}$.

3 ALGORITHM

We aim for an interpretable and disentangled representation of the controllable and uncontrollable latent features. We define controllable features as the characteristics of the MDP that are directly affected by any action $a \in \mathcal{A}$, such as the position of the agent. The uncontrollable features are those attributes that are not or only marginally affected by the actions. We show that the proposed disentanglement is possible within a single encoder by designing losses and gradient propagation through two separate parts of the latent representation. Additionally, we aim to enforce structure in our representation by means of forward prediction in latent space.

We consider environments where the high-dimensional states are pixel inputs. These pixel inputs are subsequently encoded into a latent state $z_t = (z^c, z^u) \in \mathcal{Z} \in \mathbb{R}^{n_c} + \mathbb{R}^{n_u}$, with the superscripts c and u representing the controllable and uncontrollable features, and the superscripts n_c and n_u
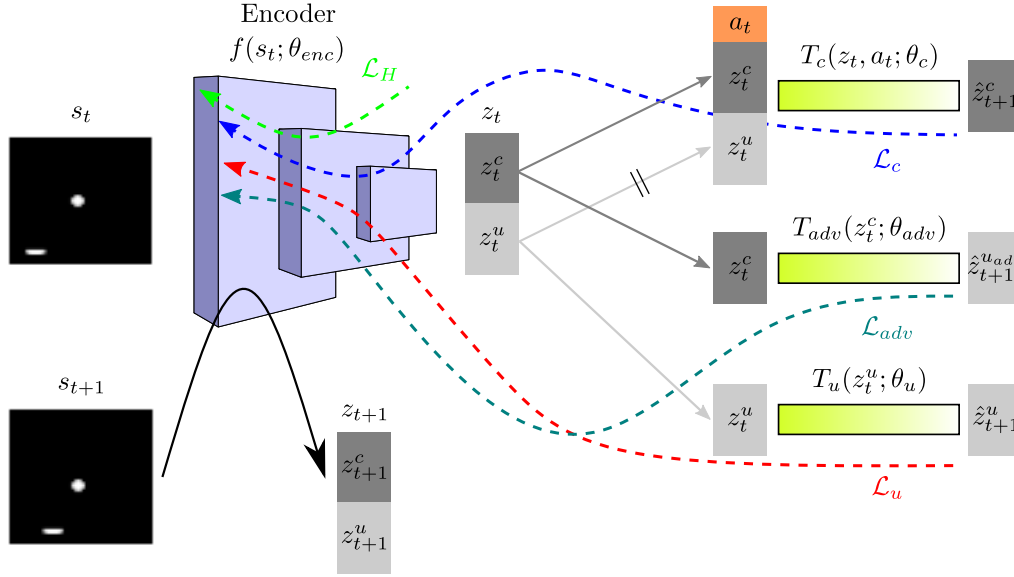
Figure 2: Overview of the disentangling architecture, with dashed lines representing gradient propagation and green rectangles representing parameterized prediction functions.

representing their respective dimensions. The compression is done by means of a convolutional encoder parameterized by θ_{enc} according to:

$$z_t = (z_t^c, z_t^u) = f(s_t; \theta_{enc}). \tag{1}$$

An overview of the proposed algorithm is illustrated in Fig. 2 and the details are provided hereafter. In this section, all losses and transitions are given under the assumption of a continuous abstract representation and a deterministic transition function. The algorithm could be adapted by replacing the losses related to the internal transitions with generative approaches (in the context of continuous and stochastic transitions) or a log-likelihood loss (in the context of stochastic but discrete representations).

3.1 CONTROLLABLE FEATURES

To isolate controllable features in the latent representation, z_t^c is used to make an action-conditioned forward prediction in latent space. In the context of a continuous latent space and deterministic transitions, z^c is updated using a mean squared error (MSE) forward prediction loss $\mathcal{L}_c = \left| \hat{z}_{t+1}^c - z_{t+1}^c \right|^2$, where $\hat{z}_{t+1}^c$ is the action-conditioned forward prediction of the parameterized function $T_c(z, a; \theta_c) : \mathcal{Z} \times \mathcal{A} \to \mathcal{Z}$:

$$\hat{z}_{t+1}^c = T_c(z_t, a_t; \theta_c) + z_t^c \tag{2}$$

and the prediction target z_{t+1}^c is part of the encoder output $f(s_{t+1}; \theta_{enc})$. Note that the full latent state z_t is necessary in order to predict $\hat{z}_{t+1}^c$ (e.g. the uncontrollable features could represent a wall or other static structure that is still necessary for the prediction of the controllable features). Furthermore, the uncontrollable latent z^u is accompanied by a stop gradient to ensure that there are no controllable features present in z^u. When minimizing $\mathcal{L}_c$, both θ_{enc} and θ_c are updated, which allows shaping the representation z^c as well as learning the internal dynamics.

3.2 UNCONTROLLABLE FEATURES

To express uncontrollable features in the latent space, z_t^u is used to make a state-only (not conditioned on the action a_t) forward prediction in latent space. This enforces uncontrollable features within the uncontrollable latent space z^u, since features that are action-dependent cannot be accurately predicted with the preceding state only. Following a prediction, z^u is then updated using a MSE forward prediction loss $\mathcal{L}_u = \left| \hat{z}_{t+1}^u - z_{t+1}^u \right|^2$, with $\hat{z}_{t+1}^u$ defined as:

$$\hat{z}_{t+1}^u = T_u(z_t^u; \theta_u) + z_t^u \tag{3}$$

and $T_u(z; \theta_u) : \mathcal{Z} \to \mathcal{Z}$ representing the parameterized prediction function. The target z_{t+1}^u is part of the output of the encoder $f(s_{t+1}; \theta_{enc})$. When minimizing $\mathcal{L}_u$, both θ_{enc} and θ_u are updated. In this way the loss $\mathcal{L}_u$ drives the latent representation z^u, which is conditioned on θ_{enc}, to only represent parts of the input state s_t that are not conditioned on the action a_t.

3.3 Avoiding Predictive Representation Collapse

Minimizing a forward prediction loss in latent space $\mathcal{Z}$ is prone to collapse (Francois-Lavet et al., 2019; Gelada et al., 2019), due to the convergence of $\mathcal{L}_c$ and $\mathcal{L}_u$ when $f(s_t; \theta_{enc})$ is a constant $\forall \ s_t \in \mathcal{S}$. To avoid representation collapse when using forward predictors, an entropy loss is used to enforce sufficient diversity in the latent representation:

$$\mathcal{L}_{H_1} = exp\big(-C_d\big\|z_t - \bar{z}_t\big\|_2\big) \tag{4}$$

where C_d represents a constant hyperparameter and $\bar{z}_t$ is a 'negative' batch of latent states z_t, which is obtained by shifting each position of latent states in the batch by a random number between 0 and the batch size. In the random maze environment, an additional entropy loss is added to further diversify the controllable representation:

$$\mathcal{L}_{H_2} = exp\big(-C_d\big\|z_t^c - \bar{z}_t^c\big\|_2\big) \tag{5}$$

where z_t^c is obtained from a single trajectory. The resulting entropy loss $\mathcal{L}_H$ thus consists of $\mathcal{L}_{H_1} + \mathcal{L}_{H_2}$. The total loss propagating through the encoder is now:

$$\mathcal{L}_{enc} = \mathcal{L}_c + \mathcal{L}_u + \mathcal{L}_H \tag{6}$$

3.4 Guiding Feature Disentanglement with Adversarial Loss

When using a controllable latent space $z^c \in \mathbb{R}^x, x \in \mathbb{N}$, where $x > g$, with g representing the number of dimensions needed to portray the controllable features, some information about the uncontrollable features in the controllable latent representation might be present (see Appendix D.2). This is due to the non-enforcing nature of $\mathcal{L}_c$, since the uncontrollable features are equally predictable with or without the action. To ensure that no information about the uncontrollable features is kept in the controllable latent representation, an adversarial component is added to the architecture in Fig. 2. This is done by updating the encoder with an adversarial loss $\mathcal{L}_{adv}$ and reversing the gradient (Ganin et al., 2016). The adversarial loss is defined as

$$\mathcal{L}_{adv} = \big|\hat{z}_t^{u_{adv}} - z_t^u\big|^2 \tag{7}$$

with

$$\hat{z}_t^{u_{adv}} = T_{adv}(z_t^c; \theta_{adv}), \tag{8}$$

where $\hat{z}_t^{u_{adv}}$ is the uncontrollable prediction of the parameterized function $T_{adv}(z; \theta_{adv}) : \mathcal{Z} \to \mathcal{Z}$ and z_t^u is the target.

Intuitively, since the parameters of $T_{adv}(z; \theta_{adv})$ are being updated with $\mathcal{L}_{adv}$ and the parameters of $f(s; \theta_{enc})$ are being updated with $-\mathcal{L}_{adv}$, the prediction function can be seen as the discriminator and the encoder can be seen as the generator (Goodfellow et al., 2014). The discriminator tries to give an accurate prediction of the uncontrollable latent z^u given the controllable latent z^c, while the generator tries to counteract the discriminator by removing any uncontrollable features from the controllable representation. When using an adversarial loss, the total loss propagating through the encoder consists of:

$$\mathcal{L}_{enc} = \mathcal{L}_c + \mathcal{L}_u + \mathcal{L}_H - \mathcal{L}_{adv}. \tag{9}$$

Where the minus term in $-\mathcal{L}_{adv}$ represents a gradient reversal to the encoder. Note that the losses are not scaled, as this did not prove to be necessary for the experiments conducted.

3.5 DOWNSTREAM TASKS

By disentangling a latent representation in a controllable and an uncontrollable part, one obtains human-interpretable features. This is a key aspect, however it is also important to test whether a notion of human interpretability equals good performance.

This is examined by training an RL agent on the learned and subsequently frozen latent representation; the action a_t is chosen following an ϵ-greedy policy, where a random action is taken with a probability ϵ, and with $(1 - \epsilon)$ probability the policy $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(z, a; \theta)$ is evaluated, where $Q(z, a; \theta)$ is the Q-network trained by Deep Double Q-Learning (DDQN) (van Hasselt et al., 2016). The Q-network is trained with respect to the target Y_t:

$$Y_t = r_t + \gamma Q(z_{t+1}, \arg\max_{a \in \mathcal{A}} Q(z_{t+1}, a; \theta); \theta^-). \tag{10}$$

With γ representing the environment's discount factor and θ^- the target Q-network's parameters. The target Q-network's parameters are updated as an exponential moving average of the original parameters θ according to: $\theta^-_{k+1} = (1 - \tau)\theta^-_k + \tau\theta_k$, where subscript k represents a training iteration and τ represents a hyperparameter controlling the speed of the parameter update. The resulting DDQN loss is defined as $\mathcal{L}_Q = \left| Y_t - Q_{(z_t, a; \theta)} \right|^2$.

4 EXPERIMENTS

In this section, we showcase the disentanglement of controllable and uncontrollable features on three different environments: (i) a quadruple maze environment, (ii) the catcher environment and (iii) a random maze environment. The first environment is relatively simple and is used to showcase the algorithm's ability to disentangle low-dimensional latent representations. The catcher environment examines a setting where the uncontrollable features are not static, and the random maze environment is used to showcase disentanglement in a more complex distribution of environments, followed by the application of downstream tasks with learning and planning. The base of the encoder is derived from Tassa et al. (2018) and consists of two convolutional layers, followed by a fully connected layer for low-dimensional latent representations or an additional CNN for a higher-dimensional latent representation such as a feature map. For the full network architectures, we refer the reader to Appendix C. In all environments, the encoder $f(s; \theta_{enc})$ is trained from a buffer $\mathcal{B}$ filled with transition tuples (s_t, a_t, r_t, s_{t+1}) using randomly sampled actions $a_t \in \mathcal{A}$.

4.1 MAZE ENVIRONMENT

The maze environment consists of an agent and a selection of four distinct, handpicked wall architectures. The environment's state is provided as pixel observations $s_t \in \mathbb{R}^{1 \times 48 \times 48}$, where an action moves the agent by 6 pixels in each direction (up, down, left, right) except if this direction is obstructed by a wall. There is no reward ($r_t = 0 \;\; \forall \;\; (s_t, a_t) \in (\mathcal{S}, \mathcal{A})$) and there is no terminal state.

We take a two-dimensional controllable representation ($z^c \in \mathbb{R}^2$) and a one-dimensional uncontrollable representation ($z^u \in \mathbb{R}^1$). The remaining hyperparameters and details can be found in Appendix A. The experiments are conducted using a buffer $\mathcal{B}$ filled with random trajectories from the four different basic maze architectures. The encoder's parameters are updated using $\mathcal{L}_{enc}$ in Equation 6 with $\mathcal{L}_H = \mathcal{L}_{H_1}$. After 50k training iterations, a clear disentanglement between the controllable (z^c) and uncontrollable (z^u) latent representation can be seen in Fig. 1. One can observe that the encoder is updated so that the one-dimensional latent representation z^u learns different values that define the type of wall architecture. A progression to this representation is provided in Appendix D.1.

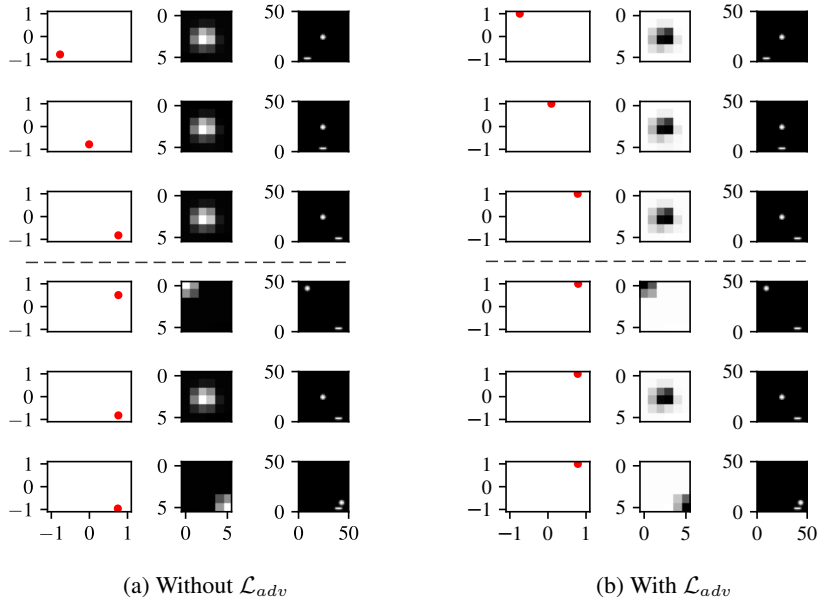(a) Without $\mathcal{L}_{adv}$ (b) With $\mathcal{L}_{adv}$

Figure 3: Visualization of the latent feature disentanglement in the catcher environment after 200k training iterations, with $z_t = f(s_t; \theta_{enc}) \in \mathbb{R}^2 + \mathbb{R}^{6 \times 6}$. In (a) and (b), the left column shows z_t^c, the middle column is z_t^u and the right column is the pixel state s_t. The dashed lines separate observations where the ball position or the paddle position are kept fixed for illustration purposes. z^c clearly tracks the agent position while z^u tracks the falling ball. In b), note that even when having a two-dimensional controllable state (only 1 is needed, see Appendix A), the adversarial loss makes sure that distinct ball positions have a negligible effect on z^c, even when the high-level features of the agent and the ball might be hard to distinguish.

4.2 CATCHER ENVIRONMENT

As opposed to the maze environment, the catcher environment has uncontrollable features that are non-stationary. The ball is dropped in a random place at the top of the environment and is falling irrespective of the actions, while the paddle position is directly modified by the actions. The environment's states are defined as pixel observations s_t of size $\mathbb{R}^{1 \times 51 \times 51}$. At each time step, the agent moves left or right by 3 pixels. Since we are only doing unsupervised learning, we consider the context where there is no reward ($r_t = 0 \ \forall \ (s_t, a_t) \in (\mathcal{S}, \mathcal{A})$) and an episode ends whenever the ball reaches the paddle or the bottom of the frame.

We take $z^c \in \mathbb{R}^2$ and $z^u \in \mathbb{R}^{6 \times 6}$. To test disentanglement, z^c is of a higher dimension than needed since the paddle (agent) only moves on the x-axis and would therefore require only one feature (see Appendix D.2 for the simpler setting with $z^c \in \mathbb{R}^1$). To show disentanglement, the redundant dimension of z^c should not or negligibly have information about z^u. The encoder's parameters are updated using $\mathcal{L}_{enc}$ in Equation 9 with $\mathcal{L}_H = \mathcal{L}_{H_1}$. After 200k iterations, a selection of state observations s_t and their encoding into the latent representation $z = (z^c, z^u)$ can be seen in Fig. 3.

4.3 RANDOM MAZE ENVIRONMENT

The random maze environment is similar to the maze environment from Section 4.1, but consists of a large distribution of procedurally generated mazes with complex wall structures. The environment's state is provided as pixel observations $s_t \in \mathbb{R}^{1 \times 48 \times 48}$, where an action moves the agent by 6 pixels in each direction. We consider $z^c \in \mathbb{R}^2$ and $z^u \in \mathbb{R}^{6 \times 6}$. This environment tests the generalization properties of a disentangled latent representation, as there are over 25 million possible maze architectures, corresponding to a probability of less then $4 \cdot 10^{-8}$ to sample the same maze twice. Note because z^c is 2-dimensional, results with and without adversarial loss are in practice extremely close. After 50k training iterations, the latent representation $z = (z^c, z^u)$ shows an interpretable disentanglement between the controllable and the uncontrollable features (See Fig. 4). Instead of
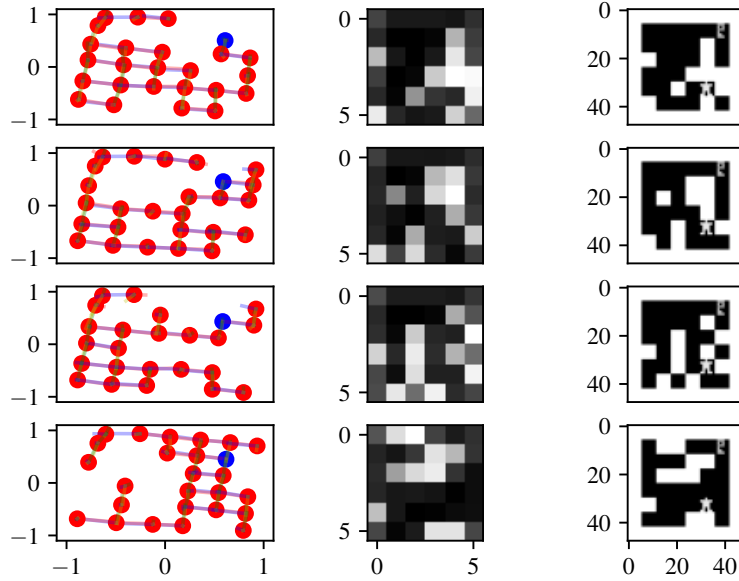
Figure 4: The disentangled latent state representations of four different random maze observations. The left column represents the controllable latent representation $z_t^c \in \mathbb{R}^2$, with the current state as blue dots, the remaining states in the same maze as red dots and the estimated latent transitions for each action as colored lines. The middle column represents the uncontrollable latent representation $z_t^u \in \mathbb{R}^{6 \times 6}$ and the right column is the original pixel state $s_t \in \mathbb{R}^{1 \times 48 \times 48}$.

using a scalar to 'describe' the uncontrollable features z^u (see Fig. 1), using a feature map for z^u allows training an encoding that provides an interpretable shape of the actual wall architecture.

Using an Inverse Predictor   An inverse prediction loss is often referred to when speaking of controllable features (Jonschkowski & Brock, 2015; Pathak et al., 2017; Badia et al., 2020). A single-step inverse prediction loss is defined as:

$$\hat{a}_t = I(z_t^c, z_{t+1}^c, z_t^u; \theta_{inv}).\tag{11}$$

Here, $\hat{a}_t$ is the predicted action and $I(z_t^c, z_{t+1}^c, z_t^u; \theta_{inv}) : \mathcal{Z} \to \mathcal{A}$ is the inverse prediction network. To see whether an inverse predictor can generate structured, controllable representations in the random maze environment, we replace the action-conditioned forward predictor with an inverse predictor, so that z^c is no longer updated with $\mathcal{L}_c$ but with $\mathcal{L}_{inv}$ (See Appendix A.3 for details on $\mathcal{L}_{inv}$).

The resulting representation can be seen in Fig. 6a. It seems that using $\mathcal{L}_{inv}$, causes an absence of interpretable structure in the controllable latent representation z_t^c. Furthermore, there is a less precise disentanglement between the controllable and uncontrollable features, as differences can be seen in z_t^c when encoding equal agent positions as pixel states s_t. In addition, an inverse predictor does not allow forward prediction in latent space, which can be used for planning as shown hereafter.

Reinforcement Learning   In order to verify whether a human-interpretable disentangled latent encoding is informative enough for downstream tasks, we formalize the random maze environment into an MDP with rewards. The agent acquires a reward r_t of -0.1 at every time step, except when it finds the key in the top right part in which case it acquires a positive reward of 1. The episode ends whenever the positive reward is obtained or a total of 50 environment steps have been taken. At an episode end, a new random wall structure is generated, and the agent starts over in the bottom left section of the maze (See Fig. 5).

Training is done for 500k iterations with the hyperparameters found in Appendix B. To see whether an interpretable disentangled latent representation is useful for RL, we compare different scenarios of (pre)training. The resulting performances are compared in Fig. 5. We find that a disentangled
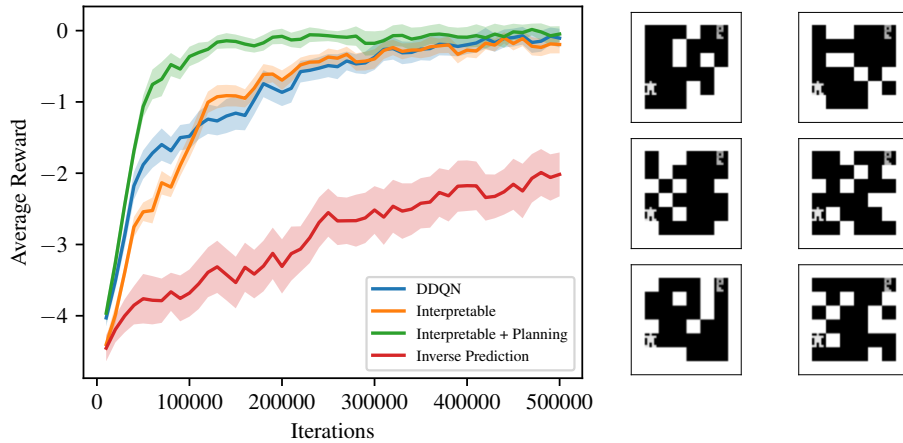
7

Figure 5: Performance of different (pre)trained representations on the random maze environment, measured as a mean (full line) and standard error (shaded area) over 5 seeds. The 'Interpretable' setting uses an encoder pre-trained with 50k random iterations to acquire a representation as in Fig. 4, after which the encoder is frozen and a Q-network is trained on top with DDQN for 500k iterations. The 'Interpretable + Planning' curve is similar to the 'Interpretable' setting, but uses a planning algorithm with a depth of 3 instead of an ϵ-greedy policy. The 'DDQN' setting uses an encoder trained end-to-end with DDQN for 500k iterations and the 'Inverse Prediction' setting is equal to the 'Interpretable' setting but has an encoder pre-trained with $\mathcal{L}_{inv}$ instead of an $\mathcal{L}_c$ for the controllable latent features z^c. On the right, a subset of random mazes used in the reward evaluation is shown.

structured representation is suitable for downstream tasks, as it achieves comparable performance to training an encoder end-to-end with DDQN for 500k iterations. Although their performance is similar, Fig. 6b shows that an encoder trained with DDQN often loses any form of interpretability. Moreover, we show in Fig. 5 that an attempt to isolate controllable features with an inverse prediction loss leads to poor downstream performance in the random maze environment.

Planning As seen in Fig. 4, after pre-training with the unsupervised losses, an interpretable disentangled representation with the corresponding agent transitions is obtained. Due to this disentanglement of the controllable and uncontrollable features, we can for instance employ prior knowledge that the uncontrollable features are static, and run a planning algorithm in the controllable latent space only. The planning algorithm used is derived from Oh et al. (2017), and is used to successfully plan only in the controllable subset of the latent representation z^c, while using the same input for z^u regardless of planning depth. More details on the planning algorithm can be found in Appendix A.2. It can be observed that even when planning with a relatively small depth of 3, we achieve better performance than the pre-trained representation with an ϵ-greedy policy and than the pure DDQN-updated encoder.

5 RELATED WORK

Many works have focused on converting high-dimensional image inputs to a compact, abstract representation to improve generalization and performance. Learning this representation can make use of auxiliary tasks in addition to the pure RL objectives (Jaderberg et al., 2017). One way to ensure a meaningful latent space is to implement architectures that require a pixel reconstruction loss such as a variational (Kingma & Welling, 2014; Higgins et al., 2017) or a deterministic (Yar, 2021) autoencoder. Others combined basic pixel reconstruction with latent planning (Hafner et al., 2019; 2021) or prediction (Francois-Lavet et al., 2019; Gelada et al., 2019). Although reconstruction losses prevent latent collapse and ensure a rich latent space, it also facilitates the reconstruction of task-irrelevant noise, thus possibly keeping irrelevant features in the latent space.

More closely related to our work is the work by Thomas et al. (2017), which connects individual latent features to independently controllable states in a maze using a reconstruction loss and a selectivity loss. The work by Francois-Lavet et al. (2019) visualizes the representation of an agent and its
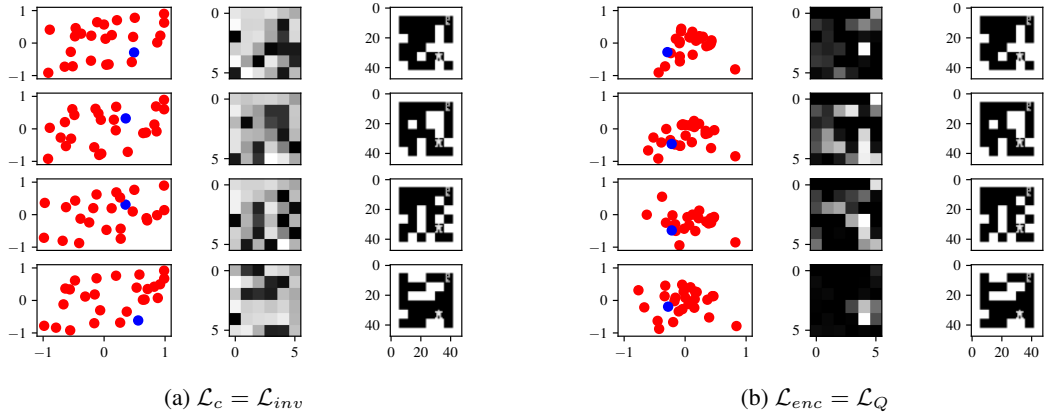
(a) $\mathcal{L}_c = \mathcal{L}_{inv}$ (b) $\mathcal{L}_{enc} = \mathcal{L}_Q$

Figure 6: Visualization of the latent representation when substituting the action-conditioned forward-prediction loss $\mathcal{L}_c$ for an inverse-prediction loss $\mathcal{L}_{inv}$ for the 50k iterations of pre-training (a) and when end-to-end updating the encoder with only the Q-loss $\mathcal{L}_Q$ from DDQN for 500k iterations (b). In both figures, the left column represent the controllable representation z^c, the middle column represents the uncontrollable representation z^u and the right column represents the original pixel state s. Evidently, the controllable representation in (a) lacks disentanglement and interpretability. Furthermore, the representation in (b) seems to have very little structure at all.

transitions in a maze environment, but does not disentangle the agent state in its controllable and uncontrollable parts, which limits the interpretability analysis and does not allow simplifications during planning. The work by Kipf et al. (2020) uses an object-oriented approach to isolate different controllable features, using graph neural networks (GNN's) and a contrastive forward prediction loss. A deeper study of predictive losses by Rakelly et al. (2021) shows the limitations and benefits of different predictive losses in control-dominant environments, albeit in a stochastic mutual information (MI) setting. Slightly related are also the works by Pathak et al. (2017); Badia et al. (2020), who try to focus on the controllable features of an environment with inverse-prediction losses and use these features to guide exploratory behaviour. Lastly, sharing similarity in terms of the separation of the latent representation, Fu et al. (2021) use a reconstruction-based adversarial architecture that divides their latent representation into reward relevant and irrelevant features.

6 CONCLUSION AND DISCUSSION

We have developed an algorithm that disentangles controllable and uncontrollable features in a single encoder architecture, while showing the potential use of this for downstream learning and planning. Disentanglement of controllable and uncontrollable features in the latent representation of high-dimensional MDPs can be achieved by propagating an action-conditioned forward prediction loss and a state-only forward prediction loss through distinct sections of the latent representation. Additionally, an entropy loss and an optional adversarial loss were used to respectively avoid collapse and further disentangle the latent representation. We have showed that these kinds of representations are human-interpretable and are able to work well with reinforcement learning. Furthermore, we showed that an action-conditioned forward predictor can be preferred as compared to an inverse predictor in terms of isolating controllable features in the representation. Additionally, by employing forward prediction in latent space, we were able to successfully plan in the controllable representation while leveraging the properties of the environment. In particular, we kept z^u static regardless of planning depth in the context of a distribution of randomly generated mazes.

Nevertheless, as environments become more complex, interpretability will be harder to enforce due to the increasing number of latent dimensions needed. Specific benefits can also be obtained with a specific design of the encoder architecture. Future work is needed to disentangle an encoder's latent representation into minimal, structured features, and finding more algorithmic benefits of this disentanglement in more complex environments. For example, in the context of safety, a disentangled interpretable representation could allow incorporating latent state constraints in a planning algorithm.

REFERENCES

Improving Sample Efficiency in Model-Free Reinforcement Learning from Images. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2021.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, and Charles Blundell. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations, ICLR*, 2020.

Vincent Francois-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined Reinforcement Learning via Abstract Representations. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2019.

Xiang Fu, Ge Yang, Pulkit Agrawal, and Tommi Jaakkola. Learning task informed abstractions. In *International Conference on Machine Learning, ICML*, 2021.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, Victor Lempitsky, Urun Dogan, Marius Kloft, Francesco Orabona, Tatiana Tommasi, and al Ganin. *Domain-Adversarial Training of Neural Networks*. In *Journal of Machine Learning Research*, volume 17, 2016.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning, ICML*, 2019.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. In *Advances in Neural Information Processing Systems, NIPS*, 2014.

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

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations, ICLR*, 2021.

Irina Higgins, Arka Pal, Andrei A. Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving Zero-Shot Transfer in Reinforcement Learning. In *International Conference on Machine Learning, ICML*, 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations, ICLR*, 2017.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Journal of Autonomous Robots*, 39(3), 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations, ICLR*, 2015.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations, ICLR*, 2014.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive Learning of Structured World Models. In *International Conference on Learning Representations, ICLR*, 2020.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. In *International Conference on Learning Representations, ICLR*, 2021.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems, NIPS*, 2020a.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *37th International Conference on Machine Learning, ICML*, 2020b.

Kuang Huei Lee, Ian Fischer, Anthony Z Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in RL. In *Advances in Neural Information Processing Systems, NIPS*, 2020.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value Prediction Network. In *Advances in Neural Information Processing Systems, NIPS*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW*, 2017.

Kate Rakelly, Abhishek Gupta, Carlos Florensa, and Sergey Levine. Which Mutual-Information Representation Learning Objectives are Sufficient for Control? In *Advances in Neural Information Processing Systems, NIPS*, 2021.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. In *International Conference on Learning Representations, ICLR*, 2021.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*, 2018.

Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently Controllable Factors. *arXiv preprint arXiv:1708.01289*, 2017.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI*, 2016.

A   ADDITIONAL MATERIAL

A.1   ABLATION OF THE ENTROPY SCALAR

Without using a pixel reconstruction loss, the entropy loss $\mathcal{L}_H$ is crucial in avoiding the trivial solution for any latent forward predictor (Francois-Lavet et al., 2019; Gelada et al., 2019). The entropy scalar that regulates the $\mathcal{L}_H$ however remains the most influential hyperparameter. When C_d is chosen too low, the representation collapses or remains in a compact cluster. On the other hand, when C_d is chosen too high, unnecessary shapes are formed to enforce large individual latent distances. Two ablations of the entropy scalar C_d are shown in Fig. 7.
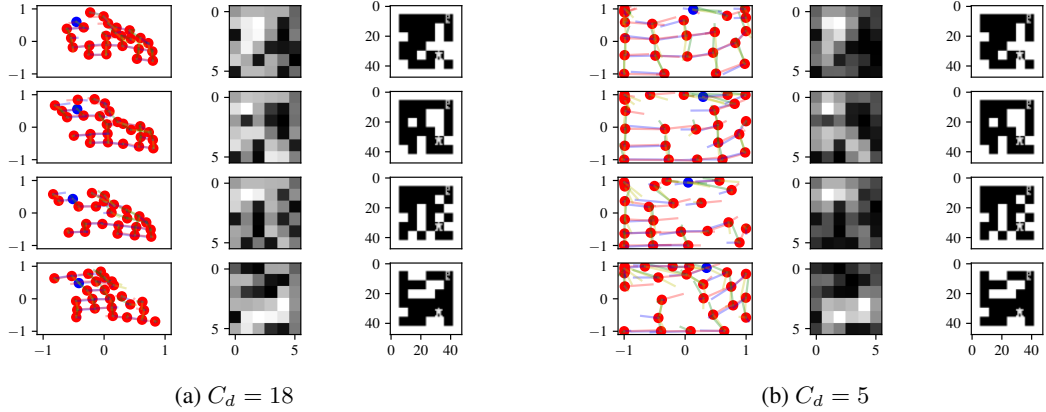

(a) $C_d = 18$

(b) $C_d = 5$

Figure 7: Ablation of the hyperparameter C_d, where a higher value of C_d enforces less entropy in the representation, while a lower value of C_d especially pushes the controllable features z^c towards shapes that ensure large distances between samples.

A.2   PLANNING

We use a planning algorithm derived from Oh et al. (2017); Francois-Lavet et al. (2019), where we employ d-step planning as:

$$\hat{Q}^d((\hat{z}_t^c, z^u), a) = \begin{cases} P((\hat{z}_t^c, z^u), a; \theta_r) + \Gamma((\hat{z}_t^c, z^u), a; \theta_\gamma) \max_{a' \in \mathcal{A}^*} \hat{Q}^{d-1}((\hat{z}_{t+1}^c, z^u), a'), & \text{if } d > 0 \\ Q((\hat{z}_t^c, z^u), a; \theta), & \text{if } d = 0 \end{cases} \quad (12)$$

$$Q_{plan}^D((\hat{z}_t^c, z^u), a) = \sum_{d=0}^{D} \hat{Q}^d((\hat{z}_t^c, z^u), a) \quad (13)$$

Where $P(s_t, a; \theta_r) : \mathcal{Z} \times \mathcal{A} \to \mathcal{R}$ represents the reward predictor and $\Gamma(s, a; \theta_\gamma) : \mathcal{Z} \times \mathcal{A} \to \gamma$ represents the discount value predictor. The action is chosen by taking the argmax of $Q_{plan}^D((\hat{z}_t^c, z^u), a)$. Note in the results from Section 4.3, we are only forward predicting in the controllable latent space z^c, and that z^u remains a fixed value regardless of planning depth. This is possible by making use of the prior knowledge of the maze environments together with a disentangled controllable and uncontrollable latent representation.

A.3   INVERSE PREDICTION

A common single-step inverse prediction is defined as:

$$\hat{a}_t = f(s_t, s_{t+1}) \quad (14)$$

where $\hat{a}_t$ is the predicted action and $f(s_t, s_{t+1})$ represents an arbitrarily structured function. In the random maze environment, we use a parameterized inverse predictor which predicts in latent space:

$$\hat{a}_t = I(z_t^c, z_{t+1}^c, z_t^u; \theta_{inv}) \tag{15}$$

Where $I(\cdot; \theta_{inv}) \in \mathcal{I} : \mathcal{Z} \to \mathcal{A}$ is a parameterized inverse prediction function. Note that we only need an arbitrary timestep for z^u, since we assume it is a static feature in our environment. Since we have 4 actions, we use the 4-dimensional logit output $\hat{a}_t$ to calculate the inverse prediction loss $\mathcal{L}_{inv}$ as:

$$S(\hat{a}_i) = \frac{\exp(\hat{a}_i)}{\sum_{j=1}^{n_a} \exp(\hat{a}_j)}, \quad \mathcal{L}_{inv} = -\sum_{i=1}^{n_a} a_i \log(S(\hat{a}_i)) \tag{16}$$

Here, n_a is the number of actions, $S(\hat{a}_i)$ represents the softmax operator and a_i is the actual action, given as a 0 or 1 truth label. This is more commonly known as the Cross-Entropy loss computation.

B EXPERIMENT DETAILS

The Pytorch framework was used for all experiments, as well as the Adam optimizer (Kingma & Ba, 2015). We employ a batch size of 32 tuples (s_t, a_t, r_t, s_{t+1}) for every update.

Simple Maze The replay buffer $\mathcal{B}$ is filled with 5k transitions from each of the four wall architectures. The transitions are collected by the agent following a random policy. The learning rate for the encoder is $5 \cdot 10^{-5}$, for the action-conditioned forward predictor $1 \cdot 10^{-3}$ and for the uncontrollable forward predictor $5 \cdot 10^{-5}$. The entropy scalar C_d is set to 15.

Catcher The replay buffer $\mathcal{B}$ is filled with 25k transitions. The transitions are collected by the agent following a random policy. A new random maze is created after 50 time-steps or when the reward is acquired. The learning rate for the encoder is $2 \cdot 10^{-5}$, for the action-conditioned forward predictor $4 \cdot 10^{-5}$ and for the uncontrollable forward predictor $1 \cdot 10^{-5}$. When using the adversarial loss, we use a learning rate of $1 \cdot 10^{-3}$ for the adversarial predictor. The entropy scalar C_d is set to 5.

Random Maze The replay buffer $\mathcal{B}$ is filled with 50k transitions, representing around 1000 maze architectures. The transitions are collected by the agent following a random policy. The learning rates used are equal to those of the catcher environment; for the encoder $2 \cdot 10^{-5}$, for the action-conditioned forward predictor $4 \cdot 10^{-5}$ and for the uncontrollable forward predictor $1 \cdot 10^{-5}$. After freezing the encoder, we train the action-conditioned forward predictor for an additional 250k iterations on the same 50k transitions in the buffer $\mathcal{B}$. For updating the Q-network with DDQN, we use a learning rate of $1 \cdot 10^{-4}$, and a τ of 0.02. The entropy scalar C_d is set to 13. When using planning, we employ a learning rate of $5 \cdot 10^{-5}$ for the reward and discount prediction networks.

Entropy Loss For the catcher and random maze environment, given that z^c is 1 or 2-dimensional, and z^u is a 36-dimensional feature map, we alleviate dimensional mismatch when calculating the entropy loss in 4. This is done by taking a random subset of 15 out of 36 feature values in z^u for every batch.

C NETWORK ARCHITECTURE

We use the same base encoder for all experiments, made up of 2 convolutional layers of 32 channels each, with a kernel size of 3 and stride 2, except for the final layer which has stride 1. Both convolutional layers have a Rectified Linear Unit (ReLU) nonlinear activation.

In the quadruple maze environment, the output of the base convolutional encoder is flattened and used as an input to a single linear layer with 3 outputs ($z^c + z^u$) and a hyperbolic tangent (tanh) activation function.

In the catcher and random maze environments, we use the following encoder head to extract the uncontrollable features; the base convolutional layers are followed by a single convolutional layer

with 32 channels, a kernel size of 4 and a stride of 1. This layer is followed by a ReLU activation function and an AveragePool layer with an output size of 6. For the controllable features, we flatten the output of the base convolutional encoder and use this as an input to a linear layer with 200 neurons and a tanh activation function. This layer is followed by another linear layer with n_c neurons and a tanh activation function.

The transition and prediction models all have the same structure, with linear layers of 32-128-128-32-x neurons where x is the output dimension in line with the predicted feature's dimension. The linear layers all have tanh activation functions except for the final output. Only the action-conditioned transition predictor of the random maze environment has larger layer sizes, with linear layers of 128-512-512-128-2, to account for slightly more complicated transitions. The DQN network used is of size 128-512-512-128-4, with an output value corresponding to each possible action.

D ADDITIONAL FIGURES

D.1 QUADRUPLE MAZE

(a) 1k iterations (b) 2k iterations (c) 5k iterations

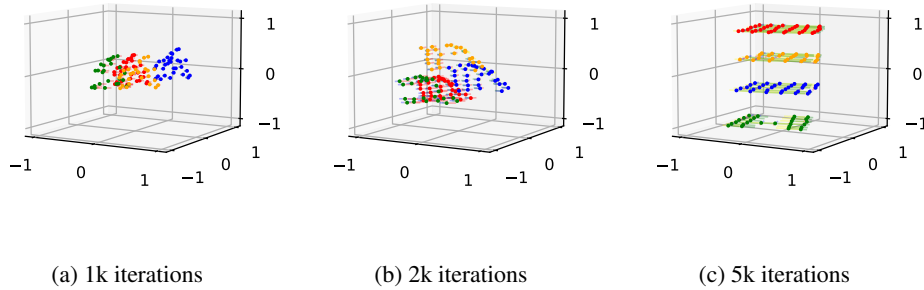Figure 8: Progression of the separation of the controllable z^c (x and y-axis) and uncontrollable z^u (z-axis) features in the maze environment.

D.2 CATCHER

(a) $z^c \in \mathbb{R}^1$ without $\mathcal{L}_{adv}$ (b) $z^c \in \mathbb{R}^2$ without $\mathcal{L}_{adv}$
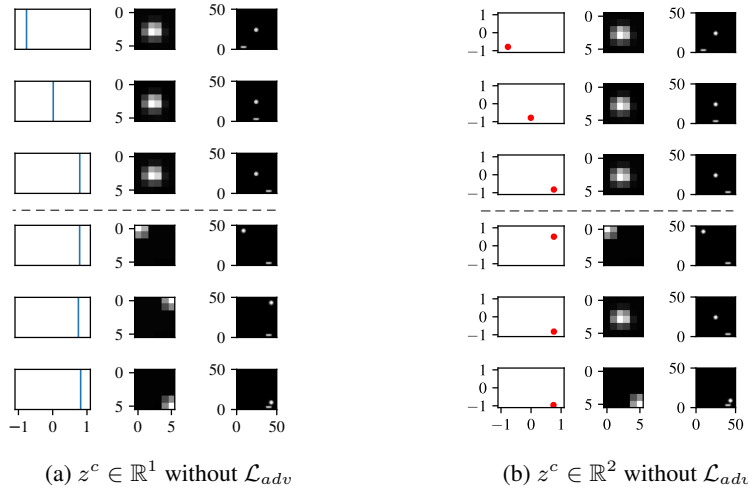
Figure 9: Comparison of training the representation for the catcher environment with either 1 or 2-dimensions for the controllable representation z^c. When using more dimensions for z^c than needed, it can be observed that some information of the ball position can be present in z^c.