REINFORCEMENT LEARNING WITH PROTOTYPICAL REPRESENTATIONS

Denis Yarats
NYU & FAIR
denisyarats@cs.nyu.edu

Rob Fergus
NYU

Alessandro Lazaric
FAIR

Lerrel Pinto
NYU

ABSTRACT

Learning effective representations in image-based environments is crucial for sample efficient Reinforcement Learning (RL). Unfortunately, in RL, representation learning is confounded with the exploratory experience of the agent – learning a useful representation requires diverse data, while effective exploration is only possible with coherent representations. Furthermore, we would like to learn representations that not only generalize across tasks but also accelerate downstream exploration for efficient task-specific training. To address these challenges we propose **Proto-RL**, a self-supervised framework that ties representation learning with exploration through prototypical representations. These prototypes simultaneously serve as a summarization of the exploratory experience of an agent as well as a basis for representing observations. We pre-train these task-agnostic representations and prototypes on environments without downstream task information. This enables state-of-the-art downstream policy learning on a set of difficult continuous control tasks. Finally, we open-source our code at https://github.com/denisyarats/proto.

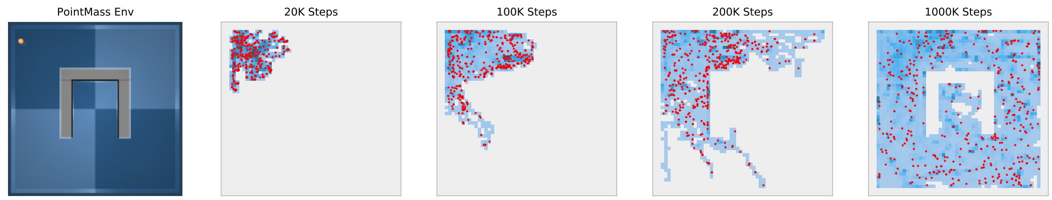
1 INTRODUCTION

Reinforcement Learning (RL) with rich visual observations has proven to be a recipe for success in a variety of domains ranging from gameplay Mnih et al. (2013); Silver et al. (2016) to robotics Levine et al. (2015). A crucial ingredient for successful image-based RL is to learn an encoder that maps the high-dimensional input to a compact representation capturing the *latent* state of the environment. Standard RL methods can then be applied using the latent representation to efficiently learn policies. Unfortunately, representation learning in RL poses several challenges.

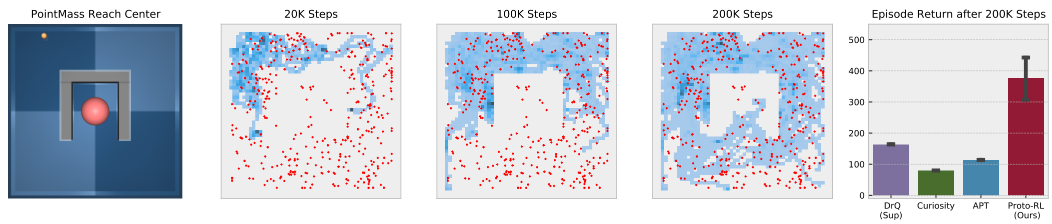
First, fitting encoders using the scarce supervisory signal from rewards alone is sample inefficient and leads to poor performance. Prior work Srinivas et al. (2020); Yarats et al. (2019) addresses this problem by leveraging self-supervised techniques alongside standard image-based RL, which leads to more robust and effective representations. However, such techniques are limited to settings that have access to task-specific rewards during representation learning.

A second, more fundamental challenge is that in the context of RL, representation learning is intimately connected to the exploration of the environment and vice-versa Sekar et al. (2020); Liu & Abbeel (2021). The data observed by the agent is non-stationary and depends on the regions of state space covered during exploration. If exploration is ineffective, the latent space produced by the encoder cannot properly characterize all parts of the environment, degrading performance for downstream tasks. Conversely, the exploration strategy cannot be defined directly on input images since no algorithm would be able to exhaustively explore all possible images. Hence, a representation accurately capturing the latent state of the environment is needed so that the agent can distinguish novel latent states from those already visited and focus exploration on the former. This leads to a chicken and egg problem, where learning useful representations requires diverse data, while effective exploration is only possible with coherent representations.

Finally, a desirable property of latent representations is to generalize across tasks defined in the same environment. This requires the representation to support a wide range of policies. Furthermore, the representation should also facilitate exploration for new tasks through the organization inherent in the latent space. Recent approaches to representation learning in RL Srinivas et al. (2020); Laskin et al.



(a) Evolution of the state-visitation distribution and prototypes during the task-agnostic pre-training phase of Proto-RL.



(b) Evolution of the state-visitation distribution during the downstream RL phase of Proto-RL for the Reach Center task.

Figure 1: An example of **Proto-RL** running on the pixel-based U-maze pointmass environment with **(a)** task-agnostic pre-training, followed by **(b)** downstream RL. **(a)**: task-agnostic exploration and representation learning stage. The state-visitation distribution is shown in blue, which converges to uniform coverage with sufficient steps. Red points depict the prototypes via closest states in the embedding space. **(b)**: subsequent application to the Reach Center task with sparse reward. Note the rapid exploration of the environment facilitated by the pre-trained prototypes (and embedding function). Proto-RL discovers the goal location in only 200k steps, while other methods struggle to solve the task. The experiment details are provided in Appendix D.

(2020); Yarats et al. (2021) are often effective for a specific task but the representations obtained are intrinsically tied to it and often perform poorly on small variations to the objective. Furthermore, when attempting a new task, even with a representation that generalizes well, the agent is still required to explore the environment during policy learning.

In this work, we address the three challenges described above through **Proto-RL**, a framework for image-based RL that learns self-supervised visual representations without access to task-specific rewards. Concretely, we consider the few-shot unsupervised RL setting, which consists of two distinct phases. In the first phase, the agent explores an environment in a task-agnostic fashion to learn its visual representations. Then in the second phase, given the learned representations, the agent is required to solve a downstream task with as few environment interactions as possible. Such a setup evaluates the agent's ability to quickly solve a new task without any prior knowledge of it.

During the first phase Proto-RL learns an encoder to embed visual observations in a low-dimensional latent space, along with a set of prototypical embeddings, which we refer to as *prototypes* (Asano et al., 2020; Caron et al., 2021) that form the basis of this latent space. To effectively explore the environment in the absence of task rewards, Proto-RL trains a policy that maximizes intrinsic reward measured by particle-based entropy (Singh et al., 2003). The latent embeddings and prototypes are trained together on observations from the exploration policy. The policy receives intrinsic reward computed using current prototypes to encourage the visitation of unexplored regions of state space. During the second phase Proto-RL uses the pre-trained encoder along with the prototypes to accelerate RL for downstream tasks.

Figure 1 illustrates the behavior of Proto-RL in an image-based navigation task. In particular, it shows the effectiveness of the unsupervised exploration strategy to thoroughly cover the state space, thus providing a diverse enough dataset for representation learning. Proto-RL also returns discrete prototypes that are evenly spread over the state space and are used to improve exploration during the downstream stage. An accurate representation of the latent state, together with efficient prototype-based exploration, leads Proto-RL to achieve state-of-the-art performance.

To summarize, this paper makes the following contributions: (i) We propose a novel task-agnostic pre-training scheme that learns an embedding function, along with a set of prototypical representations directly from visual observations; (ii) we demonstrate the ability of the learned representations and prototypes to generalize to unseen downstream tasks from the DeepMind Control Suite (Tassa et al., 2018), with significant improvements over current state-of-the-art methods; (iii) we show that the prototypes and learned representation enables efficient downstream exploration, especially in sparse reward settings.

2 Related Work

In this section we provide a brief description on the most relevant work and ideas that Proto-RL builds on top of.

Self-supervised learning in Computer Vision (CV) Self-supervision has proven to be an effective technique to learn representations from large amounts of unlabeled data Vincent et al. (2008); Doersch et al. (2015); Wang & Gupta (2015); Noroozi & Favaro (2016); Zhang et al. (2017); Gidaris et al. (2018). Several creative ideas have been used to self-supervise such as video tracking Wang et al. (2019), augmentation prediction Chen et al. (2020), and puzzle solving Noroozi & Favaro (2016) among others. Such pre-trained representations provide a strong initialization for downstream finetuning for tasks such as image classification Chen et al. (2020); Hénaff et al. (2019); Wu et al. (2018); He et al. (2020). Proto-RL is partly inspired by SwAV (Caron et al., 2021), where prototypical representations are learned through contrastive losses van den Oord et al. (2018) that encourage consistency across random augmentations of the input. However, unlike SwAV that learns on a stationary dataset of images, Proto-RL operates in dynamic environments in an RL setting that inherently produces non-stationary data distributions for learning.

Representation learning in RL To enable sample efficient RL from pixels, several researchers have taken inspiration from the successes of representation learning in computer vision and looked at learning coherent latent representations alongside RL. SAC-AE (Yarats et al., 2019), SLAC Lee et al. (2019a), demonstrated how auto-encoders can be used to learn representations that improve RL. Following this, CURL (Srinivas et al., 2020), SPR Schwarzer et al. (2020), ATC Stooke et al. (2020) used losses that encourage consistency across random observational augmentations to further improve sample-efficiency. Model-based RL has also looked at learning these representations from predictive losses Hafner et al. (2018; 2019); Yan et al. (2020); Finn et al. (2015); Agrawal et al. (2016). We note that these works in general focus on learning continuous representations of the environment through interactive experience without explicitly encouraging exploration. In contrast, Proto-RL not only learns representations on interactive experience, but also uses prototypes for effective exploration.

Exploration and Intrinsic Motivation in RL A fundamental problem in RL is exploring the state space of the underlying MDP, especially in cases where the reward is sparse or absent. Approaches that tackle this problem are generally task-agnostic and exploit various inductive biases that correlate positively with efficient exploration. Prior approaches include using state visitation counts (Bellemare et al., 2016; Ostrovski et al., 2017), curiosity-driven exploration (Pathak et al., 2017a), distilling random networks Burda et al. (2018), hindsight relabeling (Andrychowicz et al., 2017), state visitation entropy maximization (Hazan et al., 2019; Mutti et al., 2020; Liu & Abbeel, 2021), ensemble disagreement (Sekar et al., 2020), among others. Proto-RL builds on these ideas and focuses on exploration by maximizing the entropy of the state visitation distribution (Hazan et al., 2019). However, in contrast to prior work (Liu & Abbeel, 2021), Proto-RL uses prototypical representations to better estimate entropy, which improves downstream exploration.

3 Background

3.1 Task-Agnostic RL from Images

We formulate task-agnostic image-based control as an infinite-horizon partially observable Markov Decision Process (POMDP) (Bellman, 1957; Kaelbling et al., 1998) without rewards, as a tuple $\mathcal{M} = (\mathcal{O}, \mathcal{A}, P, \gamma, d_0)$, where $\mathcal{O}$ is the high-dimensional observation space (image pixels), $\mathcal{A}$ is the

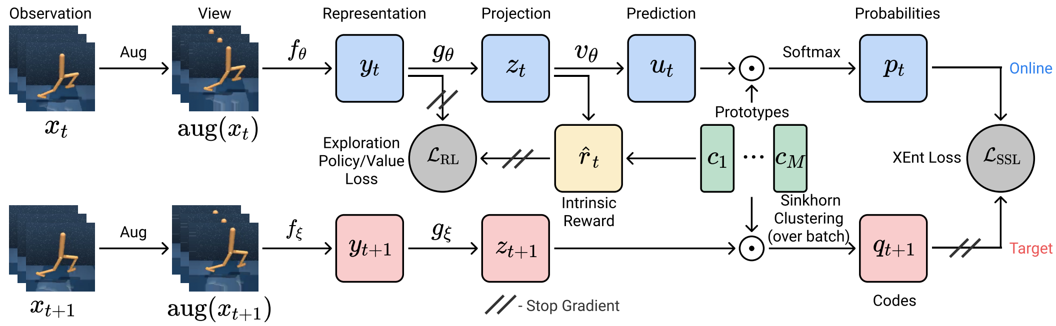


Figure 2: Proto-RL proposes a self-supervised scheme that learns to encode high-dimensional image observations $\boldsymbol{x}_t$, $\boldsymbol{x}_{t+1}$, using an encoder f_θ along with a set of prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$ that defines the basis of the latent space. Learning is done by optimizing the clustering assignment loss $\mathcal{L}_{\text{SSL}}$. To encourage exploration, prototypes are simultaneously used to compute an entropy-based intrinsic reward $\hat{r}_t$ that is maximized by the exploration agent. To decouple representation learning from the exploration task, we block the gradients of the agent's RL loss $\mathcal{L}_{\text{RL}}$ from updating the encoder and prototypes. See Section 4 for a full description.

action space, $P : \mathcal{O}^* \times \mathcal{A} \to \Delta(\mathcal{O})$ is the transition function[1] that defines a probability distribution over the next observation given the sequence of past observations and the current action, $\gamma \in [0, 1)$ is a discount factor, and $d_0 \in \Delta(\mathcal{O})$ is the distribution of the initial observation $\boldsymbol{o}_0$. Per common practice (Mnih et al., 2013), throughout the paper the task-agnostic POMDP is converted into a task-agnostic MDP (Bellman, 1957) $(\mathcal{X}, \mathcal{A}, P, \gamma, d_0)$ by stacking three consecutive previous image observations into a trajectory snippet $\boldsymbol{x}_t = \{\boldsymbol{o}_t, \boldsymbol{o}_{t-1}, \boldsymbol{o}_{t-2}\}$ and defining the corresponding state space $\mathcal{X}$ and the transition function $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$. Any policy $\pi : \mathcal{X} \to \Delta(\mathcal{A})$ induces discounted state visitation distribution $d^\pi(\boldsymbol{x}) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t d_t^\pi(\boldsymbol{x})$, where $d_t^\pi(\boldsymbol{x}) = P(\boldsymbol{x}_t = \boldsymbol{x}|\boldsymbol{x}_0 \sim d_0, \forall t' < t, \boldsymbol{a}_{t'} \sim \pi(\cdot|\boldsymbol{x}_{t'}), \boldsymbol{x}_{t'+1} \sim P(\cdot|\boldsymbol{x}_{t'}, \boldsymbol{a}_{t'}))$. Similar to Hazan et al. (2019); Lee et al. (2019b); Mutti et al. (2020), we focus on the exploratory goal of finding the policy π that maximizes the entropy $\mathbb{H}(d^\pi) = -\sum_{\boldsymbol{x}} d^\pi(\boldsymbol{x}) \log(d^\pi(\boldsymbol{x}))$ of the state visitation distribution.

3.2 Task-Specific RL from Images

In the downstream RL setup the reward-free MDP is extended with a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to form the task-specific MDP $(\mathcal{X}, \mathcal{A}, P, R, \gamma, d_0)$. The objective then is to find a policy $\pi : \mathcal{X} \to \Delta(\mathcal{A})$ to maximize the expected discounted sum of rewards $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\boldsymbol{x}_0 \sim d_0$, and $\forall t$ we have $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{x}_t)$, $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$, and $r_t = R(\boldsymbol{x}_t, \boldsymbol{a}_t)$.

3.3 Nearest Neighbor Entropy Estimation

Estimation of entropy for a distribution $p(\mathcal{X})$ defined on a q-dimensional space $\mathcal{X} \subseteq \mathbb{R}^q$ is often done via Monte Carlo using a finite set of samples $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N \sim p(\mathcal{X})$ to obtain $\hat{\mathbb{H}}_{\boldsymbol{X}}(p) = -\frac{1}{N}\sum_{i=1}^N \log p(\boldsymbol{x}_i)$. However, this estimator requires the ability to not only sample from p, but also to estimate pointwise density. This is often intractable in high-dimensional continuous spaces, such as those in image-based RL. An alternative approach is to use a non-parametric Nearest Neighbor (NN) based entropy estimator Singh et al. (2003):

$$\hat{\mathbb{H}}_{k,\boldsymbol{X}}(p) = -\frac{1}{N}\sum_{i=1}^{N} \ln \frac{k\Gamma(q/2+1)}{N\pi^{q/2}R_{i,k,\boldsymbol{X}}^q} + C_k,$$

where Γ is the gamma function, $C_k = \ln k - \frac{\Gamma'(k)}{\Gamma(k)}$ is the bias correction term, and $R_{i,k,\boldsymbol{X}} = \|\boldsymbol{x}_i - \text{NN}_{k,\boldsymbol{X}}(\boldsymbol{x}_i)\|$ is the Euclidean distance between $\boldsymbol{x}_i$ and its k^{th} nearest neighbor from the dataset $\boldsymbol{X}$, defined as $\text{NN}_{k,\boldsymbol{X}}(\boldsymbol{x}_i)$.

[1] We denote by $\mathcal{O}^*$ an arbitrarily long sequence of observations and by $\Delta(\mathcal{O})$ a distribution over the space of observations $\mathcal{O}$.

Going forward, we are only interested in the proportional estimation of entropy that simplifies the estimator:

$$\hat{\mathbb{H}}_{k,\boldsymbol{X}}(p) \propto \sum_{i=1}^{N} \ln \|\boldsymbol{x}_i - \mathrm{NN}_{k,\boldsymbol{X}}(\boldsymbol{x}_i)\|. \tag{1}$$

Here, each point $\boldsymbol{x}_i$ contributes an amount proportional to $\|\boldsymbol{x}_i - \mathrm{NN}_{k,\boldsymbol{X}}(\boldsymbol{x}_i)\|$ to the total entropy of the dataset $\boldsymbol{X}$. This estimator is shown to be asymptotically unbiased and consistent (Singh et al., 2003).

4 Proto-RL Algorithm

In this section, we provide technical details on Proto-RL. Our goal is to learn visual representations in a task-agnostic fashion through interactions with a task-agnostic POMDP (see Section 3.1 for terminology and setup). To do this, we design a self-supervised scheme that fits an encoder that embeds high-dimensional observations to low-dimensional latent states and defines an exploration strategy that allows for the discovery of diverse transitions. In Section 4.1, we describe our representation learning framework, which focuses on learning prototypes that form the basis for our visual embeddings. In Section 4.2, we describe how these prototypes yield a metric that enables entropy-based exploration. Both the representations and the exploration are learned simultaneously and are collectively referred to as Proto-RL, which is summarized in Section 4.3. Once the representations are learned, we describe how they can accelerate downstream learning of tasks in Section 4.4.

4.1 Prototypical Representation Learning

Our framework learns a visual encoder that maps pixels to continuous latent embeddings, as well as a basis within this latent space, as defined by a set of prototypical vectors. Our novel self-supervised scheme trains the encoder and prototypes simultaneously by projecting observation encodings onto clusters (prototypes) and comparing them with cluster assignment targets. These are produced by a projection of the encodings of the next observation onto the prototypes, constrained to ensure uniform prototype coverage over the dataset. Our approach draws inspiration from the recent CV approach SwAV (Caron et al., 2021), adapting these ideas to the non-stationary RL setting.

The Proto-RL framework, illustrated in Figure 2, computes representations as follows. The augmented input frames $\boldsymbol{x}_t$ are mapped to a continuous embedding $\boldsymbol{y}_t$ using the convolutional image encoder f_θ. $\boldsymbol{y}_t$ then undergoes a projection by the MLP network g_θ to produce latent vector $\boldsymbol{z}_t$, and then another MLP network v_θ to attain features $\boldsymbol{u}_t$. The final step is to project $\boldsymbol{u}_t$ into a basis defined by a set of M continuous vectors $\{\boldsymbol{c}_i\}_{i=1}^M$, which we call prototypes. This is done by using a softmax to produce $\boldsymbol{p}_t$, a probability vector over the M prototypes whose components are:

$$p_t^{(i)} = \frac{\exp(\hat{\boldsymbol{u}}_t^T \boldsymbol{c}_i/\tau)}{\sum_{k=1}^M \exp(\hat{\boldsymbol{u}}_t^T \boldsymbol{c}_k/\tau)}, \text{ where } \hat{\boldsymbol{u}}_t = \frac{\boldsymbol{u}_t}{\|\boldsymbol{u}_t\|_2}$$

and τ is the softmax temperature hyper-parameter.

Learning involves simultaneously training (1) the encoder f_θ, (2) the projector g_θ, (3) the predictor v_θ and (4) the prototype vectors $\{\boldsymbol{c}_i\}_{i=1}^M$. These form the *online* network. To optimize the *online* parameters, a *target* network is used to produce a target probability vector $\boldsymbol{q}_{t+1}$. The *target* network inputs the next augmented observation $\boldsymbol{x}_{t+1}$ and encodes it using the target encoder f_ξ to produce continuous embedding $\boldsymbol{y}_{t+1}$, then $\boldsymbol{y}_{t+1}$ is fed to the target projector g_ξ to produce latent encodings $\boldsymbol{z}_{t+1}$. These target projections is then used to compute a target probability vector $\boldsymbol{q}_{t+1}$ using the prototypes. Intuitively, the vector $\boldsymbol{q}_{t+1}$ represents the soft assignment of the target embedding to the prototypes. To ensure equal partitioning of the prototypes across all embeddings, we employ the Sinkhorn-Knopp clustering procedure (Cuturi, 2013; Caron et al., 2021), which is run over a mini-batch of embeddings. This clustering procedure constrains each prototype is assigned to the same number of samples in the batch while maintaining complete coverage. Operationally, given a batch size of B, the Sinkhorn-Knopp procedure begins with a $M \times B$ matrix with each element initialized to $\hat{\boldsymbol{z}}_{t+1,b}^T \boldsymbol{c}_m$, where $\hat{\boldsymbol{z}} = \boldsymbol{z}/\|\boldsymbol{z}\|_2$. It then iteratively produces a doubly-normalized matrix, the columns of which comprise $\boldsymbol{q}_{t+1}$ for the batch. The corresponding $\boldsymbol{p}_t$ and $\boldsymbol{q}_{t+1}$ are then used to

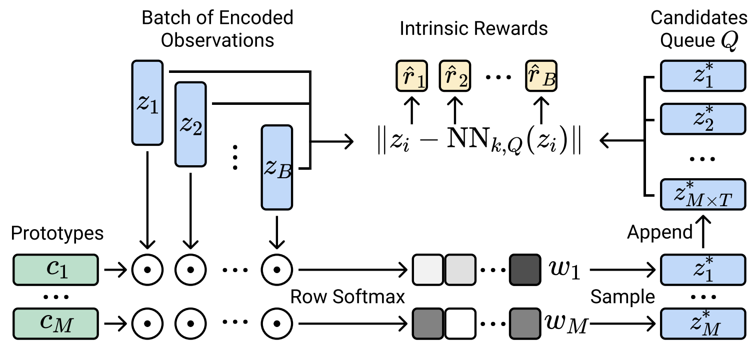


Figure 3: The entropy-based intrinsic reward used by Proto-RL. This employs a nearest-neighbor estimator (Equation (1)) computed over a set of embeddings $\boldsymbol{Q}$ that are uniformly drawn from clustering of a batch of encoded observations $\{\boldsymbol{z}_i\}_{i=1}^B$ with the current prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$. See Section 4.2 for more details.

compute a cross-entropy loss:

$$\mathcal{L}_{\text{SSL}}(\boldsymbol{p}_t, \boldsymbol{q}_{t+1}) = -\boldsymbol{q}_{t+1}^T \log \boldsymbol{p}_t.$$

Importantly, the gradient of this loss is only used to update the online network parameters (e.g. θ and $\{\boldsymbol{c}_i\}_{i=1}^M$), while being blocked in the target network. The weights of the target network ξ are instead updated using the exponential moving average of online network weights θ. Note that this update, and the use of predictor v_θ introduce an asymmetry between the two networks that prevents collapse to trivial solutions Grill et al. (2020). The pseudo-code for our framework is provided in Appendix C.

Architectures The online and target encoders f_θ and f_ξ both use the architecture from SAC-AE (Yarats et al., 2019). The online and target projectors g_θ and g_ξ are linear layers with 128 outputs. The online predictor v_θ is a 2-layer MLP with ReLU non-linearities. Proto-RL learns $M = 512$ prototypes, each parameterized as a 128-dimensional real vector.

Data The learning framework described above implements a novel contrastive scheme which compares views of two consecutive observations $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$, augmented with random image shifts Yarats et al. (2021). This differs from other representation learning for RL approaches such as CURL (Srinivas et al., 2020), which contrasts two different views of the same observation $\boldsymbol{x}_t$, and ATC (Stooke et al., 2020), which uses temporal contrast over a trajectory snippet. As mentioned in Section 3.1, the input $\boldsymbol{x}$ consists of a stack of three image frames. New data is gathered via an unsupervised exploration policy that uses current embeddings $\boldsymbol{y}_t$, projections $\boldsymbol{z}_t$ and prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$, which we detail next.

4.2 Maximum Entropy Exploration

When reward signal is absent and no assumptions about the MDP can be made, one possible intrinsic objective for the agent is to learn a policy which maximizes the entropy $\hat{\mathbb{H}}_{k,\boldsymbol{X}\sim d^\pi(\cdot)}(d^\pi)$ of the discounted state visitation distribution $d^\pi(\mathbf{x})$, per Equation (1). Although the estimator is asymptotically unbiased and consistent (Singh et al., 2003), applying it in practice poses several challenges that we address using the learned encoder and prototypes (Section 4.1).

First, estimation in the original high-dimensional image space $\mathcal{X}$ is a poor metric for measuring similarity. To this end, we estimate entropy using Euclidean distance to the k^{th} nearest neighbor in the low-dimensional learned latent space: $\hat{\mathbb{H}}_{k,\boldsymbol{Z}\sim d^\pi(\cdot)}(d^\pi) \propto \sum_{i=1}^N \ln \|\boldsymbol{z}_i - \mathrm{NN}_{k,\boldsymbol{Z}}(\boldsymbol{z}_i)\|$, where $\boldsymbol{z}_i = g_\theta(f_\theta(\boldsymbol{x}_i))$ and $\boldsymbol{Z} = \{\boldsymbol{z}_i\}_{i=1}^N$.

Second, finding the k^{th} nearest neighbor over the entire dataset $\boldsymbol{Z}$ becomes computationally expensive as the dataset grows in size. One possible solution, proposed by Liu & Abbeel (2021), is to constrain the search to a random batch $\boldsymbol{B}$ of embeddings uniformly drawn from the replay buffer $\boldsymbol{Z}$ as $\hat{\mathbb{H}}_{k,\boldsymbol{B}\sim\boldsymbol{Z}}(d^\pi)$. Empirically, this approximation leads to a high variance estimate. For example, in

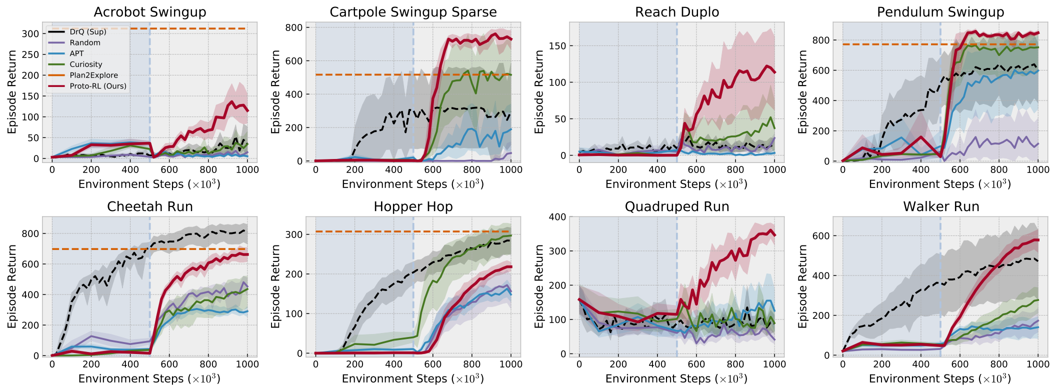


Figure 4: Single task evaluation using eight challenging environments from DeepMind Control Suite. For each method (except for DrQ and Plan2Explore), we first perform task-agnostic pretraining for 500k environment steps, before introducing task reward and training for a further 500k steps. DrQ uses task reward from the outset. Plan2Explore, being model-based, uses an intermediate methodology, described in Section 5.1. Proto-RL consistently beats the baselines and in many cases exceeds the fully supervised approach of DrQ.

a recently discovered part of the environment the state density in the buffer will be low, thus the estimated distance will be unduly large. To mitigate the problem of under-representation of novel states, we use a dataset rebalancing scheme that up-weights novel embeddings and down-weights common ones. This is done by clustering the candidates using the learned prototypes and then uniformly sampling from these clusters. To implement this for each prototype $\boldsymbol{c}_j$ we compute a softmax distribution $\boldsymbol{w}_j$ with components $w_j^{(i)} = \frac{\exp(\hat{\boldsymbol{z}}_i{}^T \boldsymbol{c}_j)}{\sum_{k=1}^B \exp(\hat{\boldsymbol{z}_k}{}^T \boldsymbol{c}_j)}$ over a batch of L_2 normalized projections $\{\hat{z}_i\}_{i=1}^B$ and then sample a constant number of candidates from this distribution. The sampled candidates are stored in a queue $\boldsymbol{Q}$ of a fixed size $M \times T$, where T candidates are used per cluster.

Finally, Proto-RL modifies an original task-agnostic transition $(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1})$ by encoding visual observations with the online encoder f_θ into embeddings $\boldsymbol{y}_t = f_\theta(\boldsymbol{x}_t)$ and $\boldsymbol{y}_{t+1} = f_\theta(\boldsymbol{x}_{t+1})$, then adding the entropy-based intrinsic reward computed using the candidate set $\boldsymbol{Q}$ as:

$$\hat{r}_t = \|\boldsymbol{z}_{t+1} - \mathrm{NN}_{k,\boldsymbol{Q}}(\boldsymbol{z}_{t+1})\|. \tag{2}$$

The proposed scheme is visualized in Figure 3.

4.3 Pretraining with Task-Agnostic RL

To collect a diverse dataset for enabling representation learning, Proto-RL simultaneously trains an exploration RL agent to optimize the intrinsic reward specified by Equation (2). The RL agent is trained on transitions $(\boldsymbol{y}_t, \boldsymbol{a}_t, \hat{r}_t, \boldsymbol{y}_{t+1})$ described above. Importantly, We block the gradients from the RL loss $\mathcal{L}_{\mathrm{RL}}$, defined in Appendix A, in order to learn task-agnostic representations and prototypes. The RL agent is implemented using SAC (Haarnoja et al., 2018).

4.4 Application to Downstream Tasks

To perform downstream RL training we (i) use the online encoder f_θ to map image observations $\boldsymbol{x}_t, \boldsymbol{x}_{t+1}$ into embeddings $\boldsymbol{y}_t, \boldsymbol{y}_{t+1}$ and (ii) augment the extrinsic reward r_t with the intrinsic reward $\hat{r}_t$, scaled by hyper-parameter α. This results into the modified transitions $(\boldsymbol{y}_t, \boldsymbol{a}_t, r_t + \alpha\hat{r}_t, \boldsymbol{y}_{t+1})$, which are then used to train a standard state-based RL algorithm. In this work, since we are interested in studying the effects of the task-agnostic representation alone, we freeze the encoder and prototypes. However, we note that finetuning representations and prototypes during downstream RL is also compatible with our framework.

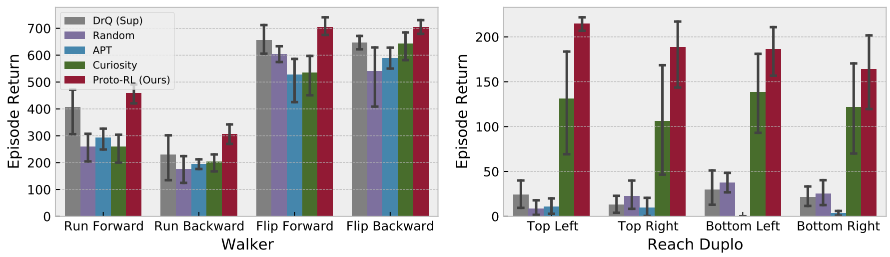


Figure 5: Multi-task evaluation using two domains from DeepMind Control Suite, with four tasks in each. We perform task-agnostic pre-training for 500k steps in each domain. The frozen representation and prototypes are then applied separately to each of the four tasks, training for additional 500k steps with the task reward. DrQ performance is measured after training for 500k steps. The results show that the representations learned by Proto-RL generalize well and enable efficient learning of multiple downstream tasks.

5 Experiments

In this section we discuss empirical results on using Proto-RL for learning visual representations. We begin by describing our experimental setup and evaluation protocols. We then use this setting to answer the following questions: (a) Does task-agnostic pre-training improve downstream task-specific RL? (b) How well do the learned representations transfer to different tasks? (c) How important is exploration during representation learning? (d) Can the pre-trained prototypes be used to improve downstream exploration?

5.1 Experimental Setup

Our agents operate in the few-shot unsupervised RL setting with two learning phases. In the task-agnostic phase the agent is allowed to interact with an environment, but it does not have access to any information about the downstream task that the agent will be asked to solve in the next phase. In the downstream RL phase, rewards associated with a task are revealed to the agent. In our experiments, agents are allowed 500k environment interactions in the task-agnostic phase, followed by 500k additional interactions with the environment in the downstream RL phase.

Environment Details We use the DeepMind Control Suite (Tassa et al., 2018), a challenging benchmark for image-based RL. Following prior work, visual observations are represented as $84 \times 84 \times 3$ pixel renderings. The episode length is 1000 for all tasks, except *Reach Duplo*, where it is 250. A fixed action repeat $R = 2$ (Hafner et al., 2019) is applied across all environments. Each agent's performance is evaluated over 10 episodes every 10000 environment steps. All figures plot the mean performance over 10 random seeds, together with ± 1 standard deviation shading.

Hyper-parameters Proto-RL is trained using Adam (Kingma & Ba, 2014) with learning rate 10^{-4} and mini-batch size of 512. The downstream exploration hyper-parameter is $\alpha = 0.2$ and the number of cluster candidates is set to $T = 4$. We use SAC implementation from Yarats & Kostrikov (2020).

Baselines To contextualize the results of Proto-RL, we compare with the following baseline algorithms:

- Random exploration: The agent is based on DrQ (Yarats et al., 2021) and it explores the environment using a random policy during the task-agnostic phase. We then freeze the learned encoder to provide representations for task-specific RL training.
- Curiosity Pathak et al. (2017b): This agent explores the environment using a curiosity-driven intrinsic motivation reward along with learning continuous visual representations using DrQ.
- APT Liu & Abbeel (2021): The agent explores the environment using an entropy-driven intrinsic motivation reward along with learning continuous visual representations. Since

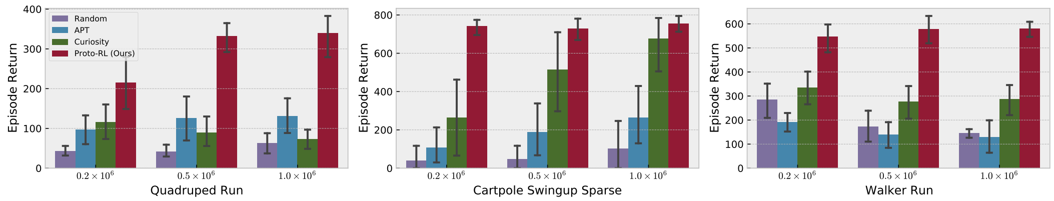


Figure 6: Varying the amount of task-agnostic pre-training on three different tasks from DeepMind Control Suite. Subsequent task-specific training (on top of the frozen representation) uses 500k steps. Proto-RL is able to explore state space sufficiently within 200k steps to learn representations that can support downstream tasks.

APT does not use prototypical representations, entropy is measured through sampling of observations from the replay buffer.

- Plan2Explore Sekar et al. (2020): Here, a model-based algorithm Dreamer (Hafner et al., 2019) is used in conjunction with Curiosity Pathak et al. (2017b) to explore the environment. However, since Plan2Explore uses model-based optimization while Proto-RL and other baselines are model-free, we only denote the final performance of Plan2Explore to avoid ambiguities in step-wise comparisons. Furthermore, Plan2Explore is provided with an estimate of the reward function of the downstream task at the end of the task-agnostic pre-training, which allows it to leverage the model to plan directly for a task-specific policy in a zero-shot manner. On the other hand, all other baselines have to learn the reward function directly in downstream RL.
- DrQ Yarats et al. (2021): Here, a state-of-the-art method for task-specific RL is trained on task-specific rewards for 1M steps to anchor the performance ranges.

The full experimental setup and details on baselines are described in Appendix B.

5.2 Task-Agnostic Pre-training

We present results on eight environments in Figure 4, with extended results on sixteen environments in Appendix F. Proto-RL significantly improves upon Random exploration and APT across all environments, while being better than Curiosity based exploration in 7/8 environments. This demonstrates that in the context of model-free RL, Proto-RL provides state-of-the-art downstream task learning. Furthermore, Proto-RL trained on 500k task-agnostic environment interactions achieves competitive performance to the model-based algorithm Plan2Explore[2] that is trained on 1M unsupervised steps, followed by the 200k fine-tuning steps with reward.

Perhaps a more exciting result is that Proto-RL trained with 500k steps of downstream RL outperforms DrQ trained on 1M steps in 6/8 environments. This demonstrates how task-agnostic representation learning can enable superior downstream RL and achieve state-of-the-art image-based RL results. Note, that these environments are indeed among the hardest image-based environments from DeepMind Control Suite (Tassa et al., 2018).

5.3 Multi-Task Generalization

As pointed out in the introduction, one desirable property of task-agnostic representations is that they can effectively generalize across different downstream tasks defined in the same environment. To highlight this ability of Proto-RL, we present results on RL training on different downstream tasks in Figure 5. After 500k steps of downstream RL, Proto-RL significantly outperforms all the baselines we compare with. We believe this ability of prototypes to accelerate downstream task learning through both better representations and exploration is key to unlock more effective and robust generalization in image-based RL tasks, as it is case in computer vision. Details about the multi-task environments are provided in Appendix E.

[2]Note that we only compare on environments reported in the original paper since the publicly released code underperforms the reported numbers.

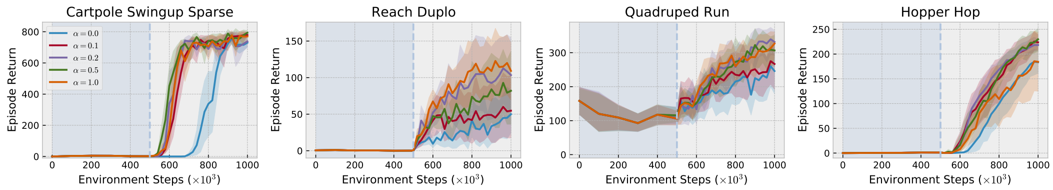


Figure 7: Evaluation regime from Figure 4 for Proto-RL but varying the balance between exploration and downstream reward using hyper-parameter α. We see that $\alpha > 0$ facilitates learning, especially in the sparse reward tasks such as *Cartpole Swingup Sparse* and *Reach Duplo*.

5.4 Efficiency of Task-Agnostic Pre-training

In the previous experiments we describe the performance of various unsupervised RL algorithms on the 500k task-agnostic steps benchmark. However, this raises a question on how many such unsupervised steps are required to learn representations that can accelerate downstream tasks. In Figure 6 we present comparisons on 200k, 500k, and 1M steps of task-agnostic training. We find that Proto-RL consistently outperforms the baselines across the various splits. Interestingly, on *Walker Run* we see that for the baselines, performance drops with increased task-agnostic training, which highlights the difficulty in learning generalizable representations without overfitting to the explored data.

5.5 Downstream Exploration

A key differentiating factor of Proto-RL compared to current relevant methods is that prototypes enable exploration even during downstream task RL. To understand the importance of this, we study of effect of the hyper-parameter α that trades off the task reward with our entropy-based intrinsic reward in Figure 7. Across all the tasks, using $\alpha = 0$ i.e., not using the prototype-driven exploration, underperforms every experiment that uses $\alpha \geq 0.1$. Notably, for sparse reward tasks like *Cartpole Swingup Sparse* and *Reach Duplo* $\alpha \geq 0.1$ is significantly better that $\alpha = 0$. This highlights the importance of using prototypes that summarize the exploratory experience in an environment. Interestingly, even without using the prototype-driven exploration, Proto-RL is able to solve all tasks with a performance that is still competitive with the baselines. This shows that the image embeddings learned during the task-agnostic pre-training are indeed effective in solving the downstream tasks.

6 Conclusion

In this paper we present Proto-RL, an unsupervised representation learning algorithm for RL. Proto-RL simultaneously learns representations and prototypes from visual inputs while exploring environments in a task-agnostic fashion. Empirically, the learned representations and prototypes enable state-of-the-art exploration and learning of downstream objectives, as well as effective generalization across multiple tasks. We believe Proto-RL brings us a step closer to “fine-tuning" in RL, a process that is commonplace in modern computer vision and natural language processing. It also opens up several directions for future research such as understanding the theoretical underpinnings of discrete representations, applications to robotics and offline RL.

REFERENCES

Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: experiential learning of intuitive physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 5092–5100, 2016.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. *arXiv preprint arXiv*, 2020.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv e-prints*, 2016.

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

Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 1957.

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

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *NIPS*, 2013.

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *CoRR*, 2015.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmassan, Stockholm, Sweden, July 10-15, 2018*, 2018.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations, 2018.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Elad Hazan, Sham M. Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration, 2019.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding, 2019.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data, 2020.

Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019a.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric P. Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *CoRR*, abs/1906.05274, 2019b. URL `http://arxiv.org/abs/1906.05274`.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, 2015.

Hao Liu and Pieter Abbeel. Unsupervised active pre-training for reinforcement learning. *openreview*, 2021. URL `https://openreview.net/forum?id=cvNYovr16SB`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv e-prints*, 2013.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. A policy gradient method for task-agnostic exploration, 2020.

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pp. 69–84. Springer, 2016.

Georg Ostrovski, Marc G. Bellemare, Aaron van den Oord, and Remi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv*, 2017a.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017b.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. *arXiv preprint arXiv*, 2020.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American Journal of Mathematical and Management Sciences*, 23 (3-4):301–321, 2003.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv*, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2018.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv e-prints*, 2015.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015.

Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019.

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3733–3742, 2018.

Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020.

Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. `https://github.com/denisyarats/pytorch_sac`, 2020.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1058–1067, 2017.

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, 2008.

APPENDIX

A EXTENDED BACKGROUND

Soft Actor-Critic The Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is an off-policy model-free RL algorithm that instantiates an actor-critic framework by learning a state-action value function Q_θ, a stochastic policy π_θ and a temperature α over a discounted infinite-horizon MDP $(\mathcal{X}, \mathcal{A}, P, R, \gamma, d_0)$ by optimizing a γ-discounted maximum-entropy objective (Ziebart et al., 2008). With a slight abuse of notation, we define both the actor and critic learnable parameters by θ. SAC parametrizes the actor policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{x}_t)$ via a $\tanh$-Gaussian defined as $\boldsymbol{a}_t = \tanh(\mu_\theta(\boldsymbol{x}_t) + \sigma_\theta(\boldsymbol{x}_t)\epsilon)$, where $\epsilon \sim \mathcal{N}(0, 1)$, μ_θ and σ_θ are parametric mean and standard deviation. The SAC's critic $Q_\theta(\boldsymbol{x}_t, \boldsymbol{a}_t)$ is parametrized as an MLP neural network.

The policy evaluation step learns the critic $Q_\theta(\boldsymbol{x}_t, \boldsymbol{a}_t)$ network by optimizing the one-step soft Bellman residual:

$$\mathcal{L}_Q(\mathcal{D}) = \mathbb{E}_{\substack{(\boldsymbol{x}_t, \boldsymbol{a}_t, \boldsymbol{x}_{t+1}) \sim \mathcal{D} \\ \boldsymbol{a}_{t+1} \sim \pi(\cdot|\boldsymbol{x}_{t+1})}}[(Q_\theta(\boldsymbol{x}_t, \boldsymbol{a}_t) - y_t)^2] \text{ and}$$
$$y_t = R(\boldsymbol{x}_t, \boldsymbol{a}_t) + \gamma[Q_{\theta'}(\boldsymbol{x}_{t+1}, \boldsymbol{a}_{t+1}) - \alpha \log \pi_\theta(\boldsymbol{a}_{t+1}|\boldsymbol{x}_{t+1})],$$

where $\mathcal{D}$ is a replay buffer of transitions, θ' is an exponential moving average of θ as done in (Lillicrap et al., 2015). SAC uses clipped double-Q learning (van Hasselt et al., 2015; Fujimoto et al., 2018), which we omit from our notation for simplicity but employ in practice.

The policy improvement step then fits the actor $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ network by optimizing the following objective:

$$\mathcal{L}_\pi(\mathcal{D}) = \mathbb{E}_{\boldsymbol{x}_t \sim \mathcal{D}}[D_{\mathrm{KL}}(\pi_\theta(\cdot|\boldsymbol{x}_t) || \exp\{\frac{1}{\alpha} Q_\theta(\boldsymbol{x}_t, \cdot)\})].$$

Finally, the temperature α is learned with the loss:

$$\mathcal{L}_\alpha(\mathcal{D}) = \mathbb{E}_{\substack{\boldsymbol{x}_t \sim \mathcal{D} \\ \boldsymbol{a}_t \sim \pi_\theta(\cdot|\boldsymbol{x}_t)}}[-\alpha \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{x}_t) - \alpha \bar{\mathcal{H}}],$$

where $\bar{\mathcal{H}} \in \mathbb{R}$ is the target entropy hyper-parameter that the policy tries to match, which in practice is set to $\bar{\mathcal{H}} = -|\mathcal{A}|$. The overall optimization objective of SAC equals to:

$$\mathcal{L}_{\mathrm{SAC}}(\mathcal{D}) = \mathcal{L}_\pi(\mathcal{D}) + \mathcal{L}_Q(\mathcal{D}) + \mathcal{L}_\alpha(\mathcal{D}).$$

We use the $\mathcal{L}_{\mathrm{SAC}}$ loss as $\mathcal{L}_{\mathrm{RL}}$ in Proto-RL.

B EXPERIMENTAL SETUP

B.1 THE DEEPMIND CONTROL SUITE SETTINGS

To benchmark our method we use the DeepMind Control Suite (DMC) (Tassa et al., 2018), a challenging set of image-based continuous control tasks. The episode length of each task is 1000 steps, except for *Reach Duplo*, where it is set to 250. Following Hafner et al. (2019), we set the action repeat hyper-parameter to 2. An environment observation $\boldsymbol{x} \in \mathcal{X}$ is constructed as a stack of 3 consecutive frames (Mnih et al., 2013), where each frame is an RGB rendering of size $3 \times 84 \times 84$ from the 0^{th} camera, except for the *Quadruped* environment, where we use the 2^{th} camera (Hafner et al., 2019), this results into a pixel tensor of size $9 \times 84 \times 84$. Finally, we divide each pixel's value by 255 to scale it down to $[0, 1]$ range.

B.2 PROTOTYPICAL REPRESENTATION LEARNING

Encoder We use the convolutional encoder architecture from SAC-AE Yarats et al. (2019) to parametrize both the online and target encoders f_θ and f_ξ. This convnet consists of four convolutional layers with 3×3 kernels and 32 channels. The `ReLU` activation is applied after each convolutional layer. We use stride to 1 everywhere, except of the first conv layer, which has stride 2. The convnet inputs tensors of dimensions $9 \times 84 \times 84$ and outputs flatten representations of size $R = 32 \times 35 \times 35 =$

39200.
Projector The online and target projectors g_θ and g_ξ are just single linear layer $39200 \to 128$ projections.
Predictor The online v_θ projector is an MLP with $128 \to 512 \to 128$ architecture and `ReLU` hidden activations.
Prototypes Proto-RL learns $M = 512$ prototypes (128-dimensional continuous vectors), where the softmax temperature is set to $\tau = 0.1$. To compute the cluster assignments target we employ the Sinkhorn-Knopp algorithm (Cuturi, 2013), which performs $n = 3$ relaxation iterations per training step.

We train the online network parameters θ and prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$ using stochastic gradient optimization with Adam (Kingma & Ba, 2014), where the learning rate is set to 10^{-4} and minibatch size to 512. The target network parameters ξ being computed as an exponential moving average of θ with momentum $\tau_{\text{enc}} = 0.05$.

B.3 Entropy-based Intrinsic Reward

Entropy is being computed in 128-dimensional latent space that is produced by the online encoder f_θ and projector g_θ. We maintain an online candidates queue Q of fixed size $M \times T = 512 \times 4 = 2048$, where each of $M = 512$ prototypes has exactly $T = 4$ candidates. The downstream exploration bonus coefficient is set to $\alpha = 0.2$.

B.4 Soft-Actor Critic Architecture

Our SAC (Haarnoja et al., 2018) implementation is based on `github.com/denisyarats/pytorch_sac` Yarats & Kostrikov (2020) with the following modifications. We add a fully-connected layer of $39200 \to 50$ with `LayerNorm` (Ba et al., 2016) activation to both actor and critic networks. We also set learning rate to 10^{-4}, minibatch size to 512, actor update frequency to 1, and critic target momentum to 0.01.

B.5 Task-Agnostic Pre-training Setup

Proto-RL simultaneously trains representations (see Appendix B.2) and exploration RL agent (see Appendix B.4) by jointly optimizing $\mathcal{L}_{\text{SSL}}$ and $\mathcal{L}_{\text{RL}}$ losses. We perform RL training in the off-policy fashion by maintainng a replay buffer of size 10^5. The exploration agent first collects 1000 seed transitions by using a random policy and stores them into the replay buffer. Further training transitions are collected by sampling actions from the exploration policy. One training update to the representations and exploration agent is performed every time a new transition is received. Given the episode's length of 1000 and fixed action repeat of 2 we thus perform 500 training updates per a training episode. In order to learn task-agnostic representations the online encoder f_θ and prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$ are only being updated with the gradients from the $\mathcal{L}_{\text{SSL}}$ loss, while the gradients from the $\mathcal{L}_{\text{RL}}$ loss are being blocked. After pre-training we fix the online encoder f_θ, online projector g_θ and prototypes $\{\boldsymbol{c}_i\}_{i=1}^M$ and prevent them from any further updates during the downstream training.

B.6 Task-Specific RL Setup

During downstream training we train a task RL agent (see Appendix B.4 for details) on the fixed representations obtained from the encoder f_θ. We also employ the pre-trained prototypes to compute intrinsic reward to combine it together with the true task reward. To ensure initial exploration we initialize the task agent's actor using the exploration actor's weights.

B.7 Full List of Hyper-Parameters

B.8 Baselines

Random We implement the Random agent baseline based on DrQ (Yarats et al., 2021). Specifically, during the task-agnostic phase the agent uses a random exploration policy to collect a replay buffer, which is used by DrQ to pre-train the convolutional encoder. During the downstream training, we

Table 1: Proto-RL list of hyper-parameters.

Parameter	Setting
Replay buffer capacity	100000
Seed steps	1000
Minibatch size	512
Action repeat	2
Discount (γ)	0.99
Optimizer	Adam
Learning rate	10^{-4}
Critic target update frequency	2
Critic target EMA momentum (τ_{Q})	0.01
Actor update frequency	2
Actor log stddev bounds	$[-10, 2]$
Encoder target update frequency	2
Encoder target EMA momentum (τ_{enc})	0.05
SAC entropy temperature	0.1
Number of prototypes (M)	512
Number of candidates per prototype (T)	4
Representation dimensionality (R)	39200
Latent dimensionality (D)	128
Softmax temperature (τ)	0.1
k in NN	3
Intrinsic reward coefficient (α)	0.2

freeze the encoder convnet and use the downstream task reward to train a DrQ policy on the fixed encoder.

Curiosity We adapt ICM (Pathak et al., 2017a) to the off-policy continuous control setting. To facilitate this, we augment DrQ (Yarats et al., 2021) with the ICM module that inputs encoded visual observations and learns forward and inverse dynamics models. The ICM module first projects the visual representations with a linear layer to 50-dimensional latent vectors. These vectors are then fed into the forward and inverse dynamics, which are parametrized by two layers MLPs with 1024 hidden units and `ReLU` nonlinearities. As per Pathak et al. (2017a), we use the forward prediction error and an intrinsic signal. We found that normalizing the curiosity reward by a running estimate of its standard deviation and then transforming with the $\log\_\text{plus}\_\text{one}$ function leads to better performance. During the task-agnostic phase the exploration agent is tasked to optimize the curiosity-based intrinsic reward. After pre-training is completed, we, again, freeze the encoder convnet and use it together with a downstream agent to optimize the target task.

APT As no original implementation is provided by Liu & Abbeel (2021), we chose to implement APT ourselves follow Liu & Abbeel (2021) as close as possible. The only difference to the original implementation is that we freeze the convolution encoder weights of APT during the downstream training to facilitate fair comparison withing our setup. This is in contrast to the setup from Liu & Abbeel (2021), where the encoder fine-tuning is allowed.

Plan2Explore We obtain results for Plan2Explore from the Table 2 in Sekar et al. (2020). We reemphasize that a direct comparison of our method to Plan2Explore is not meaningful as Sekar et al. (2020) use a different methodology and setup. Specifically, Sekar et al. (2020) allows pre-training of the reward model using the task specific rewards during the task-agnostic phase, which leaks the downstream task information. Furthermore, Plan2Explore preserves the replay buffer collected during the task-agnostic phase and uses it during the downstream training, our setup, on the other hand, completely disregards the task-agnostic transitions in the downstream stage. Finally, Plan2Explore allows further fine-tuning of the world-model during the downstream phase, while we keep the pre-trained representations fixed.

C PROTO-RL PSEUDO CODE

Algorithm 1 Pseudocode for Proto-RL training routine in a PyTorch-like style.

```
# C: M prototypes of size D (DxM)
# Q: queue of MxT candidates ((MxT)xD)
# fθ, gθ, vθ: online encoder, projector, and predictor
# fξ, gξ: target encoder and projector
# tau: momentum
# temp: temperature

# sample a minibatch of B transitions without reward from the replay buffer
# (x_t, a_t, x_{t+1}): state (Bx9x84x84), action (Bx|A|), next state (Bx9x84x84)
for (x_t, a_t, x_{t+1}) in replay_buffer:
   update_representations(x_t, x_{t+1})
   with torch.no_grad():
      r_t = compute_rewards(x_{t+1}) # compute entropy-based task-agnostic reward using the next
          state x_{t+1}
   # decouple representations from RL
   with torch.no_grad():
      y_t = fθ(x_t) # obtain representations (BxR)
      y_{t+1} = fθ(x_{t+1}) # obtain representations (BxR)
   # train exploration RL agent on an augmented minibatch of B transitions (y_t, a_t, r_t, y_{t+1})
   update_rl(y_t, a_t, r_t, y_{t+1}) # standard state-based SAC

# self-supervised representation learning routine
# x_t, x_{t+1}: state (Bx9x84x84) and next state (Bx9x84x84)
def update_representations(x_t, x_{t+1}):
   with torch.no_grad():
      C = normalize(C, dim=0, p=2) # normalize prototypes
   # online network
   x_t = aug(x_t) # random-shift view (Bx9x84x84)
   y_t = fθ(x_t) # obtain representations (BxR)
   z_t = gθ(y_t) # obtain projections (BxD)
   u_t = vθ(z_t) # obtain predictions (BxD)
   u_t = normalize(u_t, dim=1, p=2) # normalization (BxD)
   p_t = softmax(mm(u_t, C) / temp, dim=1) # assignment probabilities (BxM)
   # target network (gradient is blocked)
   with torch.no_grad():
      x_{t+1} = aug(x_{t+1}) # random-shift view (Bx9x84x84)
      y_{t+1} = fξ(x_{t+1}) # representation (BxR)
      z_{t+1} = gξ(y_{t+1}) # representation (BxD)
      z_{t+1} = normalize(z_{t+1}, dim=1, p=2) # normalization (BxD)
      q_{t+1} = sinkhorn(mm(z_{t+1}, C) / temp) # target assignments (BxM)
   # cluster assignment loss
   loss = -mean(sum(q_{t+1} * log(p_t), dim=1))
   # SGD update for online network and prototypes
   loss.backward()
   update(θ, C)
   # EMA update for the target encoder
   ξ = tau * ξ + (1 - tau) * θ

# Sinkhorn-Knopp algorithm
# S: dot products matrix (BxM)
def sinkhorn(S, n=3):
   S = exp(S).T
   S /= sum(S)
   r, c = ones(M) / M, ones(B) / B
   for _ in range(n):
      u = sum(S, dim=1)
      S *= (r / u).unsqueeze(1)
      S *= (c / sum(Q, dim=0)).unsqueeze(0)
   return (S / sum(S, dim=0, keepdim=True)).T # target assignments (BxM)

# entropy-based task-agnostic reward computation
# x: state (Bx9x84x84)
def compute_rewards(x):
   y = fθ(x) # obtain representations (BxR)
   z = gθ(y) # obtain projections (BxD)
   z = normalize(z, dim=1, p=2) # normalization (BxD)
   w = softmax(mm(z, C).T, dim=1) # candidates softmax probabilities (MxB)
   i = Categorical(w).sample() # one sample per row (Mx1)
   candidates = z[i] # select M candidates (MxD)
   enqueue(Q, candidates) # append the M candidates to the candidates Q, maintain the fixed (
       MxT) size

   # find k-nearest neighbor for each sample in z (BxD) over the candidates queue Q ((MxT)xD)
   dists = norm(z[:, None, :] - Q[None, :, :], dim=-1, p=2) # pairwise L2 distances (Bx(MxT))
       between y and Q
   topk_dists, _ = topk(dists, k=3, dim=1, largest=False) # compute topk distances (Bx3)
   r = topk_dists[:, -1:] # rewards (Bx1) are defined as L2 distances to the k-nearest
       neighbor from Q
   return r
```

D THE POINTMASS MAZE EXPERIMENT DETAILS

The U-maze environment is based on the *PointMass Easy* task from DMC (Tassa et al., 2018) with the following modifications. First, we add three walls to the MuJoCo model:

```
<default class="wall">
    <geom type="box" material="site"/>
</default>
<geom name="maze_x" class="wall" pos="-.1 0 .02" zaxis="1 0 0"  size=".02 .1 .02"/>
<geom name="maze_neg_x" class="wall" pos=".1 0 .02" zaxis="1 0 0"  size=".02 .1 .02"/>
<geom name="maze_y" class="wall" pos="0 .12 .02" zaxis="0 1 0"  size=".12 .02 .02"/>
```

We then modify initial state distribution of the point mass from being uniform across the entire $[-0.3, 0.3] \times [-0.3, 0.3]$ grid, to be uniformly distributed across a much smaller region of the state space situated in the top-left corner $[-0.3, -0.15] \times [0.15, 0.3]$. During the task-agnostic stage there is no target location and the agent explores the state space by optimizing the entropy-based intrinsic reward. During the task-specif phase, we place a target location at the center $[0, 0]$ of the grid with radius 0.07. The agent receives reward of 1 if it reaches the target location, otherwise it receives no reward. The contrived initial state distribution and sparse reward function make this task extremely hard from the exploration point of view.

E THE MULTITASK EXPERIMENT DETAILS

Walker We add four additional tasks *Run Forward*, *Run Backward*, *Flip Forward*, and *Flip Backward* to the *Walker* environment from DMC that require the agent to run forward/backward, flip forward/backward correspondingly. These tasks are similar to the *Cheetah* tasks from Plan2Explore (Sekar et al., 2020) that are implemented in `github.com/ramanans1/dm_control`.

Reach Duplo In this set of tasks the agent is required to reach the lego block that is placed in four different fixed locations: *Top Left* $[-0.09, 0.09]$, *Top Right* $[0.09, 0.09]$, *Bottom Left* $[-0.09, -0.09]$, and *Bottom Right* $[0.09, -0.09]$. These task are based on the *Reach Duplo* environment from DMC.

F Full Results for the Task-Agnostic Pre-training Experiment

We conduct the experiment defined in Section 5.2 on an extended set of 16 environments from DMC (Tassa et al., 2018) and provide them in Figure 8.

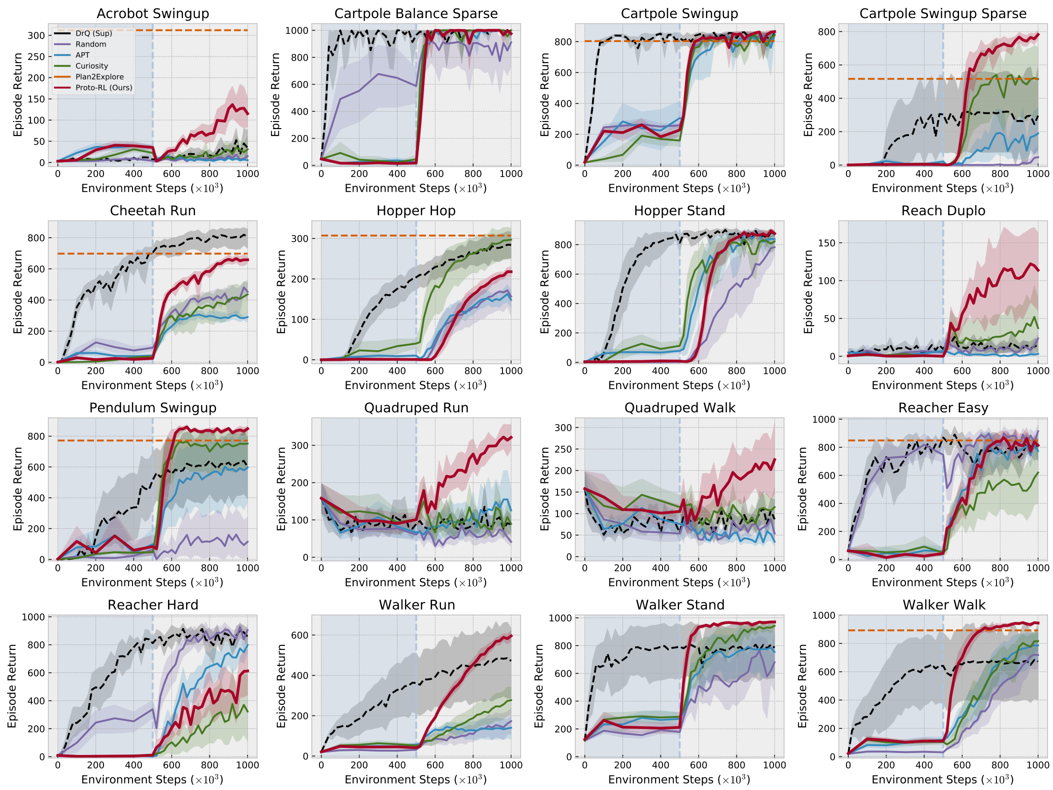


Figure 8: Single task evaluation using a full set of 16 challenging environments from DeepMind Control Suite. For each method (except for DrQ and Plan2Explore), we first perform task-agnostic pretraining for 500k environment steps, before introducing task reward and training for a further 500k steps. DrQ uses task reward from the outset. Plan2Explore, being model-based, uses an intermediate methodology. Proto-RL consistently beats the baselines and in many cases exceeds the fully supervised approach of DrQ.

G Full Results for the Efficiency of Task-Agnostic Pre-training Experiment

In Figure 9 we provide full results of the experiment from Section 5.4.

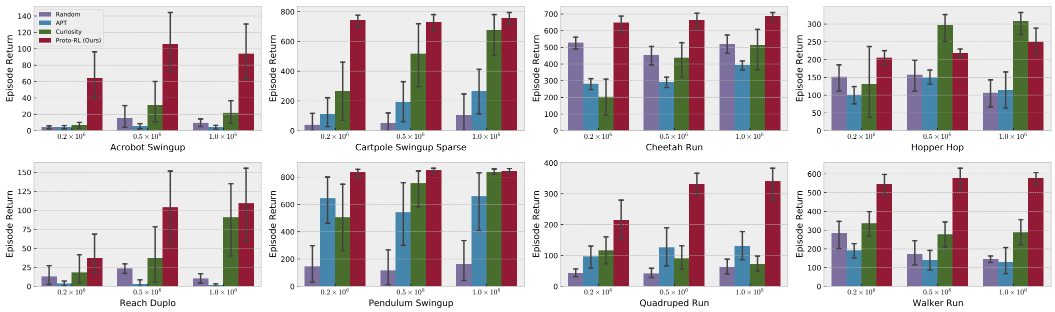


Figure 9: Varying the amount of task-agnostic pre-training on three different tasks from DeepMind Control Suite. Subsequent task-specific training (on top of the frozen representation) uses 500k steps. Proto-RL is able to explore state space sufficiently within 200k steps to learn representations that can support downstream tasks.

H Full Results for the Downstream Exploration Experiment

In Figure 10 we provide full results of the experiment from Section 5.5.

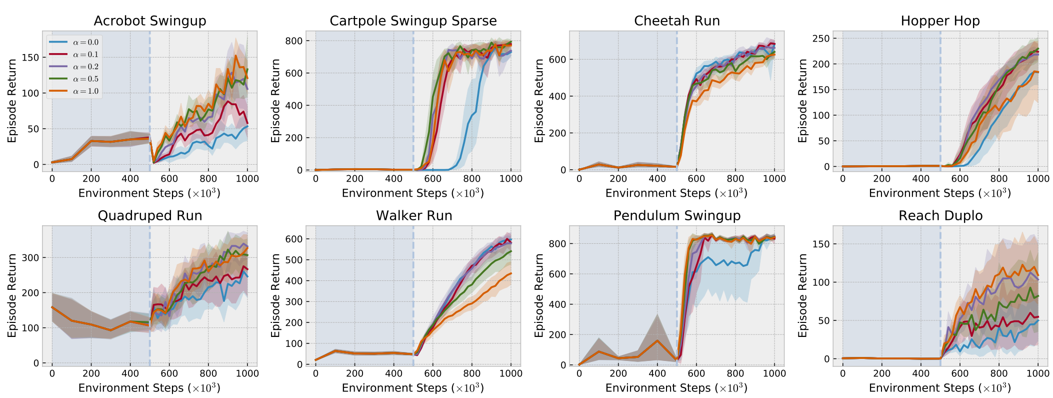


Figure 10: Evaluation regime from Figure 4 for Proto-RL but varying the balance between exploration and downstream reward using hyper-parameter α. We see that $\alpha > 0$ facilitates learning, especially in the sparse reward tasks such as *Cartpole Swingup Sparse* and *Reach Duplo*.