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.].
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 \( \mathcal{M} = (\mathcal{O}, A, P, \gamma, d_0) \), where \( \mathcal{O} \) is the high-dimensional observation space (image pixels), \( A \) is the...
Figure 2: Proto-RL proposes a self-supervised scheme that learns to encode high-dimensional image observations $x_t, x_{t+1}$, using an encoder $f_\theta$ along with a set of prototypes $\{c_i\}_{i=1}^M$ that define the basis of the latent space. Learning is done by optimizing the clustering assignment loss $L_{\text{SSL}}$. 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.

3.2 Task-Specific RL from Images

In the downstream RL setup the reward extended MDP is formed by reward function $R: S \times A \to \mathbb{R}$ to form the task-specific MDP $(X, A, R, \gamma, d_0)$. The objective then is to find a policy $\pi : X \to \Delta(A)$ to maximize the expected discounted sum of rewards $\mathbb{E}_x \sum_{t=0}^{\infty} \gamma^t r_t$, where $x_0 \sim d_0$, and $\forall t$ we have $a_t \sim \pi(\cdot|x_t), x_{t+1} \sim P(\cdot|x_t, a_t)$, and $r_t = R(x_t, a_t)$.}

3.3 Nearest Neighbor Entropy Estimation

Estimation of entropy for a distribution $p(X)$ defined on a $q$-dimensional space $X \subseteq \mathbb{R}^q$ is often done via Monte Carlo using a finite set of samples $X = \{x_i\}_{i=1}^N \sim p(X)$ to obtain $\hat{H}_X(p) = -\frac{1}{N} \sum_{i=1}^N \log p(x_i)$. 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):

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

where $\Gamma$ is the gamma function, $C_k = \ln k - \frac{\Gamma'(k)}{\Gamma(k)}$ is the bias correction term, and $R_{i,k,X} = ||x_i - NN_k,X(x_i)||$ is the Euclidean distance between $x_i$ and its $k$th nearest neighbor from the dataset $X$, defined as $NN_k,X(x_i)$. \footnote{We denote by $O^*$ an arbitrarily long sequence of observations and by $\Delta(O)$ a distribution over the space of observations $O$.}
The Proto-RL framework, illustrated in Figure 2, computes representations as follows. The augmented embedding \( q_t \) the columns of which comprise prototypes whose components are: \( \hat{z}_i^k = \frac{\exp(\hat{u}^T_i c_k / \tau)}{\sum_{k=1}^M \exp(\hat{u}^T_i c_k / \tau)} \), where \( \hat{u}_t = \frac{u_t}{\|u_t\|_2} \) and \( \tau \) is the softmax temperature hyper-parameter.

Learning involves simultaneously training (1) the encoder \( f_\theta \), (2) the projector \( q_\theta \), (3) the predictor \( v_\theta \) and (4) the prototype vectors \( \{c_i\}_{i=1}^M \). These form the online network. To optimize the online parameters, a target network is used to produce a target probability vector \( q_{t+1} \). The target network inputs the next augmented observation \( x_{t+1} \) and encodes it using the target encoder \( f_\xi \) to produce continuous embedding \( y_{t+1} \), then \( y_{t+1} \) is fed to the target projector \( q_\xi \) to produce latent encodings \( z_{t+1} \). These target projections are then used to compute a target probability vector \( q_{t+1} \) using the prototypes. Intuitively, the vector \( q_{t+1} \) 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 \( \hat{z}_i^T \delta c_m \), where \( \hat{z} = z / \|z\|_2 \). It then iteratively produces a doubly-normalized matrix, the columns of which comprise \( q_{t+1} \) for the batch. The corresponding \( p_t \) and \( q_{t+1} \) 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 $Q$ that are uniformly drawn from clustering of a batch of encoded observations $\{z_i\}_{B=1}^B$ with the current prototypes $\{c_i\}_{M=1}^M$. See Section 4.2 for more details.

compute a cross-entropy loss:

$$L_{SSL}(p_t, q_{t+1}) = -q_{t+1}^T \log p_t.$$  

Importantly, the gradient of this loss is only used to update the online network parameters (e.g. $\theta$ and $\{c_i\}_{M=1}^M$), while being blocked in the target network. The weights of the target network $\xi$ are instead updated using the exponential moving average of online network weights $\theta$. Note that this update, and the use of predictor $v_\theta$ 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 $f_\theta$ and $f_\xi$ both use the architecture from SAC-AE (Yarats et al., 2019). The online and target projectors $g_\theta$ and $g_\xi$ are linear layers with 128 outputs. The online predictor $v_\theta$ 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 $x_t$ and $x_{t+1}$, augmented with random image shifts Yarats et al. (2019). 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 $x_t$, and ATC (Stooke et al., 2020), which uses temporal contrast over a trajectory snippet. As mentioned in Section 3.1, the input $x$ consists of a stack of three image frames. New data is gathered via an unsupervised exploration policy that uses current embeddings $y_t$, projections $z_t$ and prototypes $\{c_i\}_{i=1}^M$, 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 $\hat{H}_k, X \sim d^\pi(\cdot|x)$, 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 $X$ 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: $\hat{H}_{k, Z \sim d^\pi(\cdot|x)} = \sum_{i=1}^N \ln ||z_i - \text{NN}_{k, Z}(z_i)||$, where $z_i = g_\theta(f_\theta(x_i))$ and $Z = \{z_i\}_{i=1}^N$.

Second, finding the $k$th nearest neighbor over the entire dataset $Z$ 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 $B$ of embeddings uniformly drawn from the replay buffer $Z$ as $\hat{H}_{k, B \sim Z}$. Empirically, this approximation leads to a high variance estimate. For example, in
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 \((y_t, \alpha_t, \hat{r}_t, y_{t+1})\) described above. Importantly, we block the gradients from the RL loss \(L_{RL}\), 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 \(f_\theta\) to map image observations \(x_t, x_{t+1}\) into embeddings \(y_t, y_{t+1}\) and (ii) augment the extrinsic reward \(r_t\) with the intrinsic reward \(\hat{r}_t\), scaled by hyper-parameter \(\alpha\). This results into the modified transitions \((y_t, \alpha_t, r_t + \alpha \hat{r}_t, y_{t+1})\), 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 $\pm 1$ 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 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.

Note that we only compare on environments reported in the original paper since the publicly released code underperforms the reported numbers.
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 the effect of the hyper-parameter $\alpha$ 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 than $\alpha = 0$. 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


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


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 \( Q_\theta \), a stochastic policy \( \pi_\theta \) and a temperature \( \alpha \) over a discounted infinite-horizon MDP \( (X, A, P, R, \gamma, d_0) \) by optimizing a \( \gamma \)-discounted maximum-entropy objective (Ziebart et al., 2008). With a slight abuse of notation, we define both the actor and critic learnable parameters by \( \theta \). SAC parametrizes the actor policy \( \pi_\theta(a_t|x_t) \) via a tanh-Gaussian defined as \( a_t = \tanh(\mu_\theta(x_t) + \sigma_\theta(x_t) \epsilon) \), where \( \epsilon \sim N(0, 1) \), \( \mu_\theta \) and \( \sigma_\theta \) are parametric mean and standard deviation. The SAC’s critic \( Q_\theta(x_t, a_t) \) is parametrized as an MLP neural network.

The policy evaluation step learns the critic \( Q_\theta(x_t, a_t) \) network by optimizing the one-step soft Bellman residual:

\[
L_Q(D) = \mathbb{E}_{(x_t, a_t, x_{t+1}) \sim D}[ (Q_\theta(x_t, a_t) - y_t)^2 ] \text{ and }
\]

\[
y_t = R(x_t, a_t) + \gamma[Q_\theta'(x_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|x_{t+1})],
\]

where \( D \) is a replay buffer of transitions, \( \theta' \) is an exponential moving average of \( \theta \) 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 \( \pi_\theta(a_t|x_t) \) network by optimizing the following objective:

\[
L_\pi(D) = \mathbb{E}_{x_t \sim D}[ D_{KL}(\pi_\theta(\cdot|x_t)||\exp(\frac{1}{\alpha} Q_\theta(x_t, \cdot)))].
\]

Finally, the temperature \( \alpha \) is learned with the loss:

\[
L_\alpha(D) = \mathbb{E}_{a_t \sim \pi_\theta(\cdot|x_t)}[ -\alpha \log \pi_\theta(a_t|x_t) - \alpha \hat{H}],
\]

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

\[
L_{SAC}(D) = L_\pi(D) + L_Q(D) + L_\alpha(D).
\]

We use the \( L_{SAC} \) loss as \( L_{RL} \) 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 \( x \in X \) 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 0th camera, except for the Quadruped environment, where we use the 2nd 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 \( f_\theta \) and \( f_\xi \). This convnet consists of four convolutional layers with \( 3 \times 3 \) 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 \( R = 32 \times 35 \times 35 = 34900 \).
The online and target projectors $g_θ$ and $g_ξ$ are just single linear layer $39200 \rightarrow 128$ projections.

The online $v_θ$ projector is an MLP with $128 \rightarrow 512 \rightarrow 128$ architecture and ReLU hidden activations.

Proto-RL learns $M = 512$ prototypes (128-dimensional continuous vectors), where the softmax temperature is set to $τ = 0.1$. 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 $\{c_i\}_{i=1}^M$ 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 $τ_{enc} = 0.05$.

### 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 $α = 0.2$.

### 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 $39200 \rightarrow 50$ 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 $L_{SSL}$ and $L_{RL}$ losses. We perform RL training in the off-policy fashion by maintaining 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 $\{c_i\}_{i=1}^M$ are only being updated with the gradients from the $L_{SSL}$ loss, while the gradients from the $L_{RL}$ loss are being blocked. After pre-training we fix the online encoder $f_θ$, online projector $g_θ$ and prototypes $\{c_i\}_{i=1}^M$ 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replay buffer capacity</td>
<td>100000</td>
</tr>
<tr>
<td>Seed steps</td>
<td>1000</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>512</td>
</tr>
<tr>
<td>Action repeat</td>
<td>2</td>
</tr>
<tr>
<td>Discount ($\gamma$)</td>
<td>0.99</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Critic target update frequency</td>
<td>2</td>
</tr>
<tr>
<td>Critic target EMA momentum ($\tau_Q$)</td>
<td>0.01</td>
</tr>
<tr>
<td>Actor update frequency</td>
<td>2</td>
</tr>
<tr>
<td>Actor log stddev bounds</td>
<td>$[-10, 2]$</td>
</tr>
<tr>
<td>Encoder target update frequency</td>
<td>2</td>
</tr>
<tr>
<td>Encoder target EMA momentum ($\tau_{enc}$)</td>
<td>0.05</td>
</tr>
<tr>
<td>SAC entropy temperature</td>
<td>0.1</td>
</tr>
<tr>
<td>Number of prototypes ($M$)</td>
<td>512</td>
</tr>
<tr>
<td>Number of candidates per prototype ($T$)</td>
<td>4</td>
</tr>
<tr>
<td>Representation dimensionality ($R$)</td>
<td>39200</td>
</tr>
<tr>
<td>Latent dimensionality ($D$)</td>
<td>128</td>
</tr>
<tr>
<td>Softmax temperature ($\tau$)</td>
<td>0.1</td>
</tr>
<tr>
<td>$k$ in NN</td>
<td>3</td>
</tr>
<tr>
<td>Intrinsic reward coefficient ($\alpha$)</td>
<td>0.2</td>
</tr>
</tbody>
</table>

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 plus 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 with 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.

```plaintext
# 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)
        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) # 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 = 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), τ, 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))
    # compute topk distances (Bx3)
    r = topk_dists[:, -1:] # rewards (Bx3) are defined as L2 distances to the k-nearest neighbor from Q
    return r
```

17
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:

```xml
<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-specific 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](image_url)

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](image_url)

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