FADEE: Faster Adaptation for Decoupled Exploration and Exploitation

Sergio Charles
Stanford University
Department of Computer Science
sergioc1@stanford.edu

Raghav Samavedam
Stanford University
Department of Computer Science
raghavsa@stanford.edu

Arnav Joshi
Stanford University
Department of Computer Science
ajoshi21@stanford.edu

Abstract

Motivated by recent advances in decoupled meta reinforcement learning, we sought to modify and expand upon DREAM [1]. We first developed extensions of the MapGrid environment that required significantly more involved optimal exploration strategies. By benchmarking vanilla DREAM we found that a recurrent policy was necessary for task-specific exploration. This was confirmed by a number of t-distributed stochastic neighbour embeddings of final exploration policy cell states that indicated strong clusters determined by task ID. This led us to conclude that the cell state of the exploration policy had to encode a belief of the task at hand. While experiments to directly use final cell state as a trajectory embedding failed to converge, a new exploration algorithm based on the task adaptation in PEARL [2] was found to achieve optimal rewards in around as much or fewer timesteps as the DREAM based implementation. This method's latent variable was designed to explicitly reflect a belief of the task at hand, which was enough context to provide for an exploration policy that had no hidden state. To improve the robustness of the trajectory encoder, we developed a contrastive loss based on SimCLR [3], finding that it has a positive effect on exploration reward. The contrastive loss we used included a number of augmentations, such as a random starting position in the grid world environment and a randomly sampled hidden state to encode various amounts of initial prior information. Next, we developed methods for semi-supervised meta-reinforcement learning where we only have a limited number of supervised problem IDs and a large number of unsupervised problem IDs. Thus, we effectively decrease the number of fully observed problem IDs needed to do full end-to-end training. By leveraging the SimCLR approach technique, we used only a small number of supervised problem IDs and a large number of unsupervised problem IDs to train the exploration and exploitation policies. We developed an additional k-means clustering technique to infer unsupervised problem IDs, which motivated a MAML-based clustering method to adapt quickly in various unsupervised task settings. This avenue of research yielded some positive observations that we hope to work on in future work.

The implementation is publicly available on GitHub: https://github.com/sergiogcharles/fadee

1 Introduction & Objective

Reinforcement learning techniques have been deployed in several high-profile use cases – including games like Atari, Go and Chess [4, 5, 6] – and have surpassed human performance in these instances. Given that RL methods generally have poor sample complexity and unstable convergence, there is a strong incentive to try to learn many new tasks efficiently given the experiences of other tasks. This general aim is known as meta reinforcement-learning (meta-RL). Many methods in meta-RL focus on efficiently adapting a policy based on online or offline data from a rollout on a new task [7, 8, 9, 10, 11, 12].

A number of novel algorithms, however, aim to optimize methods for efficiently collecting environment-specific data to aid the RL policy in adapting to new tasks. Specifically, this involves thoroughly exploring an environment to efficiently gain the information necessary to solve a bevy of potential tasks. Unfortunately, training any policy using vanilla end-to-end RL methodologies does not directly optimize for this form of robust exploration. This is because optimal exploration involves taking actions that are only indirectly useful to achieving a goal - these kind of indirect actions are not strongly incentivized with the highly reward-driven nature of vanilla RL policies.

In DREAM, the problem of ineffective exploration is dealt with by explicitly decoupling the exploration and exploitation paradigms by separately training an exploration policy to maximize information learned about the given environment. Previous methods for optimizing an exploration policy in a decoupled setting suffer from issues regarding how their objectives are precisely defined, with their exploration policies either gathering task irrelevant data [13] or having convergence issues due to the co-dependent nature of exploration and execution [12, 14, 15, 16].

Broadly speaking, DREAM learns an encoder that turns a task-ID into a latent representation that can inform exploitation, and trains a exploration policy that learns to produce this latent representation by way of a decoder.

$$\textbf{Encoder :} F(\boldsymbol{z}|\mu) \tag{1}$$
$$\textbf{Exploration} : \pi^{\text{exp}}(\boldsymbol{a}|, \boldsymbol{h}) \tag{2}$$
$$\textbf{Decoder} : G(\boldsymbol{z}|\tau^{\text{exp}}) \tag{3}$$
$$\tag{4}$$

The exploitation policy, then, makes use of this decoded exploration trajectory to better inform its actions.

$$\textbf{Exploitation} : \pi^{\text{task}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{h}, \boldsymbol{z}) \tag{5}$$

The task ID based embeddings are trained end to end with the exploitation (also referred to as execution) policy. The exploration policy and the associated trajectory encoder are then trained to match these embeddings that must contain precisely the right information for solving their respective task.

Our goal is to improve DREAM's exploration behavior under a variety of different limitations. The contributions of our research are as follows:

1. Developed numerous didactic environments that require task-specific exploration by extending the GridWorld environment used in DREAM.
2. Determined that the recurrent hidden state used in the exploration policy encodes an increasing amount of task specific information over time.
3. Developed a refined sampling extension to the exploration policy based on PEARL [2].
4. Developed contrastive loss for improving the robustness of the exploration trajectory embedder.
5. Defined what it means to be semi-supervised in the context of DREAM, and used the contrastive loss as a way to deal with relaxing a constraint on training multiple task ID based embeddings.

2 Related Work

2.1 DREAM

Algorithm 1 DREAM meta-training

for meta-training trials **do**

Sample problem ID μ

Get exploration τ trajectory from rollout of exploration policy on the problem ID μ's environment

Train the decoder to match the encoding of the problem ID

Half the time train the encoder and execution policy end to end

Other half the time train the decoder and execution policy end to end

end for

DREAM leverages a recurrent exploration and execution policy and LSTM trajectory embedder and must train encodings for all task IDs it meta trains on. At meta-test time, the exploration policy is deployed on a new task's environment, the decoder creates a representation of that rollout, and that representation is then used by the execution policy for what hopefully should be instantaneous optimal task completion.

2.2 Contrastive Learning and SimCLR

Algorithm 2 SimCLR training algorithm for unsupervised data

Encoder Network F

for training trials **do**

Sample a batch of unsupervised data

For each data point, apply a certain transformation to it, doubling the size of the batch

For a given example x in this now expanded batch, compute the cosine similarity between its encoding and the rest of the batch

formulate this as a vector v and apply the softmax function to it.

Set the loss for that given example x to be $logv[i]$ where i is the index of the transformed version of x. This incentivizes the representation of a given data point to be more similar to its transformed counterpart relative to other data points.

Add up the losses and backpropagate on f.

end for

This unsupervised training algorithm can be combined with some labeled data and was found to extrapolate effectively. Provided the right kind of transformations, this approach can serve as a way to better cluster embedding data.

2.3 PEARL

PEARL [2] explicitly tries to do task discovery by sampling optimal policies for a wide distribution of tasks, rolling them out, collecting context data from the rollouts, and using that to gradually find the task. We seek to use this method to make the exploration policy more task specific as there could be separate optimal exploration strategies for different tasks.

3 Recurrent DDQN Policy & The Hidden State Hypothesis

Additionally, we also train a trajectory embedder directly on the last hidden / cell state associated with a trajectory rollout in a bid to capture some of the task information encoded there. Multiple architectures were tried, including directly using the hidden state itself. These approaches failed to produce trajectory embeddings needed for convergence, likely due to the fact that hidden states might

encode other information not relating to a task and that the nature of a hidden state evolves over time as the recurrent exploration trains - making it impossible to train from. Directly optimizing the hidden state by making it part of the computational graph of the execution policy's update resulted in exploding gradient instabilities.

4 Improved Didactic Environments

We wanted to begin by testing the robustness of DREAM for environments that require task-specific exploration. We define task-specific exploration as a setting in which the exploration behavior is different for different tasks. The baseline environment simply requires that, for each task, the exploration policy visit the map. Then, the exploitation policy should be able to use the learned information to effectively maximize reward.

Our first updated environment presents two different types of ways we can reach a desired goal. The first type is still the buses, the available buses go to corner states that end the episode. The second type is a singular teleportation square that is present for only a subset of tasks and always takes the agent to the right goal state. This teleportation square is not indicated from the map and must be searched for (similar to the Distracting Bus environment used in DREAM).

Our second updated environment introduces sign posts that an agent must follow to find the locations to search for the teleportation square - setting up a tough, chained exploration problem.

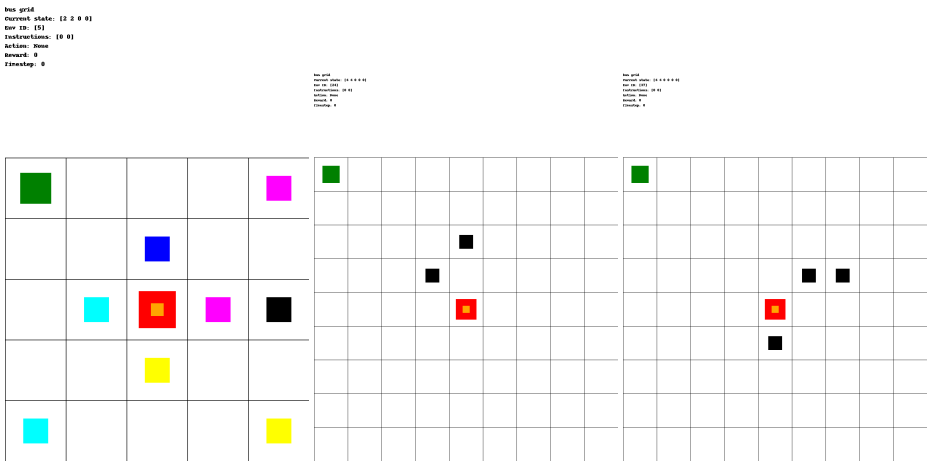

Figure 1: The first visualization is for the basic map environment. The next two visualizations are for the sign-post environment.

To compute the theoretically optimal exploitation reward, we require $\mathbb{E}[R[\pi^{\text{task}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{h}, \boldsymbol{z})]]$ – the expected exploitation reward of the optimal exploitation policy. For the first updated environment, this expectation is 0.67 and for the second it is 0.36.

5 Task-specific PEARL-like Exploration Policy for DREAM

In this section, we adapt the exploration policy of DREAM to use Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables (PEARL) task belief updates to make exploration more targeted and task-specific. We also seek to do so in a sample-efficient manner with posterior sampling, similar to PEARL.

5.1 Merging PEARL & DREAM

PEARL [2] leverages a degree of uncertainty in belief over a task to efficiently adapt to new tasks at meta-test time. Rakelly et al. estimates the belief over the task by using experiences from before-seen tasks via an off-policy RL approach. In particular, they estimate the belief of task by learning a "probabilistic latent representation of prior experience" [2].

We sample the task $\mu \sim p(\mu)$ from a uniform prior of tasks. Notationally, we define $\boldsymbol{c}_i^\mu := (\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_i')$ as a transition for task μ, whereby $\boldsymbol{c}^\mu := \{\boldsymbol{c}_i\}_{i=1}^H$ for H the finite horizon of the MDP. We adapt DREAM to use posterior sampling during meta-exploration so that we learn an exploration policy π_ϕ^{exp} that rapidly adapts to solving the task by conditioning on the context $\boldsymbol{c}^\mu$.

During meta-exploration, the belief of the current task is encoded in a latent representation $\boldsymbol{w}$. PEARL uses the data collected $\boldsymbol{c}$ from entire training episodes to infer the latent $\boldsymbol{z}$, namely approximate $p(\boldsymbol{z}|\boldsymbol{c})$. Om the other hand, during one meta-exploration episode, at each time step, we use the transitions collected from the past t time steps in the task μ to infer $p(\boldsymbol{z}|\boldsymbol{c}_{:t}^\mu)$ posterior. Then we update the policy based on the inferred latent task encoding from the posterior at each time step of the exploration episode. The reason we do this is because, in DREAM, we meta-train with trials of one exploration and one exploitation episode. At meta-test time, we are at liberty to test on many exploitation episodes.

As such, we train an inference network $G_\xi(\boldsymbol{z}|\boldsymbol{c}_{:t}^\mu)$, parameterized by ξ, that approximates $p(\boldsymbol{z}|\boldsymbol{c}_{:t})$. Thus, we distinguish the inference network $G_\xi(\boldsymbol{w}|\boldsymbol{c}_{:t})$ from the decoder $g_\omega(\boldsymbol{z}|\tau^{\text{exp}})$ and, hence, latent embeddings. It should be noted that we can coalesce the decoder network with inference network, whereby the inference network is trained on more fine-grained data. However, if we set the context $\boldsymbol{c}_{:t}$ to be the entire trajectory τ^{exp}, i.e. $t := H$, then conditioning the inference network on the trajectory is equivalent to decoding $g_\omega(\boldsymbol{z}|\tau^{\text{exp}})$. While we maintain the distinction of these two networks, one can reduce overhead by only training the inference network $G_\xi(\boldsymbol{w}|\boldsymbol{c}_{:t})$.

5.2 Variational Lower Bound & Inference Network Architecture

We train the inference network using the model-free approach to optimize the variational lower bound to model the state-action value function Q_ϕ of the exploration policy:

$$\mathbb{E}_{\mu\sim p(\mu)}\left[\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim\mathcal{B},\boldsymbol{w}\sim G_\xi(\boldsymbol{w}|\boldsymbol{c})}[Q_\phi^{\text{exp}}(s,a,\boldsymbol{w}) + \beta D_{\text{KL}}(G_\xi(\boldsymbol{w}|\boldsymbol{c}_{:t}^\mu)||p(\boldsymbol{w}))]\right] \tag{6}$$

where $p(\boldsymbol{w}) \sim \mathcal{N}(0, \boldsymbol{I})$ is the Gaussian prior, and the KL divergence term is an information bottleneck which limits mutual information between the latent task encoding $\boldsymbol{w}$ and $\boldsymbol{c}^\mu$ [2], effectively compressing $\boldsymbol{w}$ to contain only necessary information from the previous transitions. Rakelly et al. suggest that we should be able to infer the task regardless of the order in which we observe the transitions; thus, $G_\xi(\boldsymbol{w}|\boldsymbol{c}_{:t}^\mu)$ should be permutation-invariant function [2]. We factorize this model as a product of independent Gaussian factors

$$G_\xi(\boldsymbol{w}|\boldsymbol{c}_{:t}^\mu) \propto \prod_{i=1}^{t} \Psi_\xi(\boldsymbol{w}|\boldsymbol{c}_t^\mu) \tag{7}$$

where $\Psi_\xi(\boldsymbol{w}|\boldsymbol{c}_t^\mu) = \mathcal{N}\left(F_\xi^{\hat{\mu}}(\boldsymbol{c}_t^\mu), F_\xi^{\hat{\sigma}}(\boldsymbol{c}_t^\mu)\right)$. Here, F_ξ is a multi-headed feed forward neural network parameterized by ξ that outputs the parameters of the Gaussian factors $\hat{\mu}$ and $\hat{\sigma}$ when passed a context c_t^μ.

5.3 Posterior Sampling

At meta-test time, we sample a task $\mu \sim p(\mu)$ from the prior. Then we roll out the exploration policy for 1 time step $\tau_t^{\text{exp}} \sim \pi_\phi^{\text{exp}}(a_t|s_t, \tau_{:t-1}^{\text{exp}})$ and add it to the task-specific buffer B^μ. Then we update the posterior belief of the task, and rollout the exploration for one more timestep conditioned on the posterior belief, i.e. $\boldsymbol{w} \sim p(\boldsymbol{w}|\boldsymbol{c}^\mu)$. At each time step during meta-exploration, we update the belief so that we can quickly adapt to the task at hand. This task-specific exploration process is enumerated in Algorithm 1.

Meta-RL often assumes that the data distributions match at meta-train and meta-test time. Thus, since meta-testing uses on-policy data, meta-training must also use on-policy data. However, Rakelly et al. highlight that the inference network $G_\xi(\boldsymbol{w}|\boldsymbol{c}^\mu)$ and ϵ-greedy DDQN policy π_ϕ^{exp} need not be trained on the same data. That is, we use a recurrent deep dueling double Q-network to model the state-action value function $Q_\phi^{\text{exp}}(\boldsymbol{s}, \boldsymbol{a})$ on off-policy data sampled from the task replay buffer B^μ. Then we use a sampler $S_C(B^\mu)$ to sample batches of contexts to train the inference network. If we sample from the entire buffer, the distributional shift between the on-policy test data is too stark.

Thus, Rakelly et al. [2] propose sampling from recently collected data in the replay buffer, which maintains distributional similarity to the on-policy test data.

5.4 Training Exploration Policy, Inference Network, & Exploitation Policy

The exploration and exploitation policies are both parameterized as recurrent deep dueling double Q-networks. The exploration policy has Q-function approximated by $\hat{Q}_{\phi}^{\text{exp}}(\boldsymbol{s}, \boldsymbol{a}, \tau^{\text{exp}})$. Likewise, the exploitation policy's Q-function is approximated by $\hat{Q}_{\theta}^{\text{exp}}(\boldsymbol{s}, \boldsymbol{a}, \tau^{\text{task}})$. We use the encoder network $F_{\Psi}(\boldsymbol{z}|\mu)$ and decoder network $q_{\omega}(\boldsymbol{z}|\tau^{\text{exp}})$. We update the inference network $G_{\xi}(\boldsymbol{w}|\boldsymbol{c}^{\mu})$ in tandem with the DDQN $Q_{\phi}^{\text{exp}}(\boldsymbol{s}, \boldsymbol{a}, \tau^{\text{task}}, \boldsymbol{w})$. The inference network uses gradients from the exploration DDQN to perform its update, as seen in Algorithm 1. The exploration DDQN approximates the state-action value function by minimizing the following loss [1]:

$$\mathcal{L}_{\text{DDQN}}^{\text{exp}}(\phi) := \mathop{\mathbb{E}}_{\substack{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim B \\ \boldsymbol{w}\sim G_{\xi}(\boldsymbol{w}|\boldsymbol{c})}} [||\hat{Q}_{\phi}^{\text{exp}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \tau_{:t}^{\text{exp}}, \boldsymbol{w}) - \bar{Q}_{\phi'}^{\text{exp}}||^2] \tag{8}$$

for target network $\bar{Q}_{\phi'}^{\text{exp}} = r_t^{\text{exp}} + \gamma \hat{Q}_{\phi'}^{\text{exp}}(\boldsymbol{s}_{t+1}, \tau_{:t+1}^{\text{exp}}, \boldsymbol{a}_{\text{DDQN}}, \boldsymbol{w})$ where (see [1]):

$$r_t^{\text{exp}} = ||F_{\Psi}(\mu) - g_{\omega}(\tau_{:t}^{\text{exp}})||_2^2 - ||F_{\Psi}(\mu) - g_{\omega}(\tau_{:t+1}^{\text{exp}})||_2^2 - c \tag{9}$$

and

$$\boldsymbol{a}_{\text{DDQN}} := \arg\max_{\boldsymbol{a}} \hat{Q}_{\phi'}^{\text{exp}}(\boldsymbol{s}_{t+1}, \tau_{:t+1}^{\text{exp}}, \boldsymbol{a}), \boldsymbol{w}). \tag{10}$$

As such, we update the inference network $G_{\xi}(\boldsymbol{w}|\boldsymbol{c}^{\mu})$ by minimizing:

$$\mathcal{L}_{\text{infer}}(\xi) := \mathcal{L}_{\text{DDQN}}^{\text{exp}} + \beta D_{\text{KL}}(G_{\xi}(\boldsymbol{w}|\boldsymbol{c}^{\mu})||p(\boldsymbol{w})) \tag{11}$$

where we add the bottle-neck mutual information term so $\boldsymbol{w}$ only contains the necessary information about the task from the context $\boldsymbol{c}^{\mu}$.

For the exploitation DDQN policy, we sample $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', \mu, \tau^{\text{exp}})$ from the exploitation replay buffer and perform updates to minimize [1]:

$$\mathcal{L}_{\text{DDQN}}^{\text{task}}(\theta, \Psi) := \mathop{\mathbb{E}}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim B} [||\hat{Q}_{\theta}^{\text{task}}(\boldsymbol{s}, \boldsymbol{a}, F_{\Psi}(\mu)) - \hat{Q}_{\theta'}^{\text{task}}||^2] \tag{12}$$

for target network $\hat{Q}_{\theta'}^{\text{task}} := r_t^{\text{exp}} + \gamma \hat{Q}_{\theta'}^{\text{task}}(\boldsymbol{s}', F_{\Psi'}(\mu), \boldsymbol{a}_{\text{id}})$ and greedy action $\boldsymbol{a}_{\text{id}} = \arg\max_{\boldsymbol{a}} \hat{Q}_{\theta}^{\text{task}}(\boldsymbol{s}', \boldsymbol{a}, F_{\Psi}(\mu))$.

Algorithm 3 Task-specific Exploration for DREAM: Meta-Training

Require: Training tasks $\mu \sim p(\mu)$, learning rates α_1, α_2
  Initialize $G_\xi(\boldsymbol{w}|\boldsymbol{c}^\mu)$ ▷ Belief of the task encoding given context
  for meta-training trials **do**:
    Sample $\mu \sim p(\mu)$
    Encode $z \sim F_\Psi(z|\mu)$
    Initialize buffer $B^\mu = \{\}$
    for $t = 1, \ldots, H$ **do**:
      # Infer from posterior
      Sample $\boldsymbol{c}^\mu \sim S_C(B^\mu)$
      Sample $\boldsymbol{w} \sim G_\xi(\boldsymbol{w}|\boldsymbol{c}^\mu)$
      # Exploration episode
      Rollout exploration policy $\tau_t^{\text{exp}} \sim \pi_\phi^{\text{exp}}(a_t|s_t, \tau_{:t-1}^{\text{exp}}, \boldsymbol{w})$ for 1 timestep
      Add (s_t, a_t, r_t, s_t') to B^μ
      # Training inference network
      If $|B^\mu| >$ minimum buffer size:
      for $k = 1, \ldots, K$ **do**:
        Sample batch $b^\mu \sim B^\mu, c^\mu \sim S_C(B^\mu)$
        Sample from posterior $\boldsymbol{w} \sim G_\xi(\boldsymbol{w}|\boldsymbol{c}^\mu)$
        $\mathcal{L}_{\text{DDQN}}^{\mu,\text{exp}} := \mathbb{E}_{\substack{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim B \\ \boldsymbol{w}\sim G_\xi(\boldsymbol{w}|\boldsymbol{c})}}[||\hat{Q}_\phi^{\text{exp}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \tau_{:t}^{\text{exp}}, \boldsymbol{w}) - \bar{Q}_{\phi'}^{\text{exp}}||^2]$
        $\mathcal{L}_{\text{KL}}^\mu = \beta D_{\text{KL}}(G_\lambda(\boldsymbol{w}|\boldsymbol{c}^\mu)||p(\boldsymbol{w}))$
        $\xi \leftarrow \xi - \alpha \nabla_\xi (\mathcal{L}_{\text{DDQN}}^\mu + \mathcal{L}_{KL}^\mu)$
      end for
    end for
    Update $\hat{Q}_\phi^{\text{exp}}$, π_ϕ^{exp} and q_ω to minimize $I(\tau^{\text{exp}}; \boldsymbol{z})$ via rewards
    # Exploitation episode
    Every other episode, choose $\boldsymbol{z} \sim q_\omega(\boldsymbol{z}|\tau^{\text{exp}})$
    Roll out exploitation policy $\pi_\theta^{\text{task}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$
    Update π_θ^{task} and F_Ψ:

$$\max_{\Psi,\theta} \mathbb{E}_{\mu\sim p(\mu), \boldsymbol{z}\sim F_\Psi(\boldsymbol{z}|\mu)}[V^{\pi_\theta^{\text{task}}}(\boldsymbol{z}; \mu)] - \lambda I(\boldsymbol{z}; \mu)$$

  end for

5.5 Results & Analysis

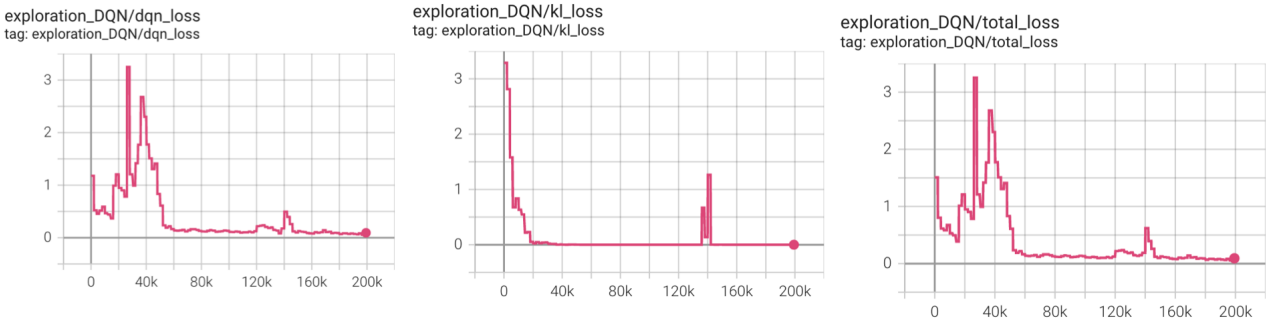

Figure 2: DQN, KL divergence, and total losses using task-specific exploration on the basic map grid environment.

Figure 3: Exploration and exploitation rewards for baseline DREAM (blue) and task-specific exploration DREAM (red) on the basic map grid environment trained for 180k episodes

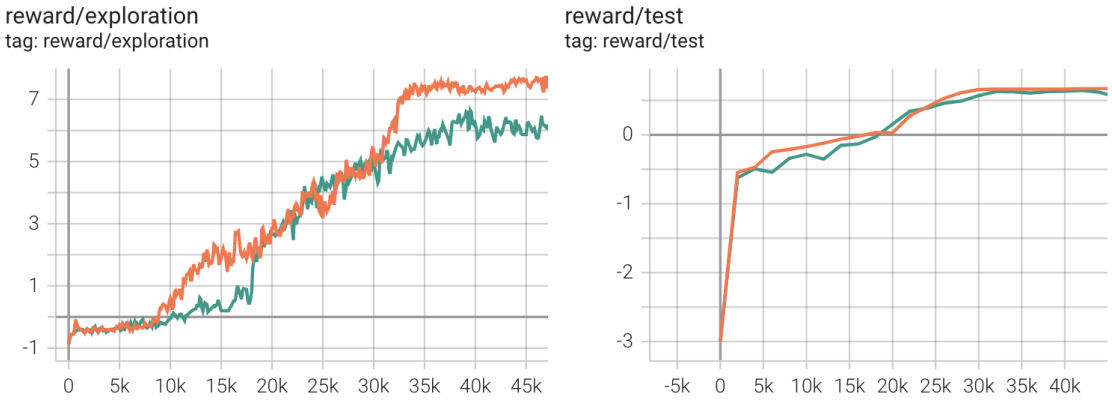

Figure 4: Exploration and exploitation rewards for baseline DREAM (green) and task-specific exploration DREAM (orange) on the teleportation square environment trained for 45k episodes

As can be seen from the figures above, our PEARL-based method with a non-recurrent exploration policy converged to the optimal reward of several different environments in about the same time DREAM did using a recurrent exploration policy. This provides evidence that the cell state for the recurrent policy must be implicitly doing some of the same work of task identification our method explicitly does, as having no recurrency makes it impossible for DREAM to solve the task-specific exploration. Our method also uses a context embedding precisely for task identification, indicating that it could lead to better results if it were used in the same way as the cell state conditioned trajectory embedding.

6 Contrastive Loss and Cell State Conditioned Trajectory Embedding

To improve the robustness of the trajectory embedder, we developed an auxillary contrastive loss to enforce consistency across representations for a given task and to push out different task embeddings from each other. In a similar vein to SimCLR [17], during meta-training of our exploration policy, we roll out the trajectory $\tau^{\text{exp}} = (s_0, a_0, r_0, s_1, \dots) \sim \pi_\phi^{\text{exp}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ from a starting state s_0 in task μ for $\boldsymbol{z} \sim F(\boldsymbol{z}|\mu)$. Then we generate a trajectory representation $\boldsymbol{\alpha} = g_\omega(\tau^{\text{exp}})$ with the trajectory embedder.

6.1 Positive Sampling Augmentations

To generate a augmented "view" of this trajectory, we start roll out a new trajectory with the exploration policy $(\tau^{\text{exp}})^+ = (s_0^+, a_0^+, r_0^+, s_1^+, \dots) \sim \pi_\phi^{\text{exp}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ from a random starting state of s_0^+ in task μ where $\boldsymbol{z} \sim F(\boldsymbol{z}|\mu)$. Then we embed this trajectory as $\boldsymbol{\alpha}^+ = g_\omega((\tau^{\text{exp}})^+)$. We want to force $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^+$ to be nearby in the trajectory embedding space because this makes the exploration and, thus, exploitation policy robust to random state initializations on the grid. In the terminology of SimCLR [17], $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^+$ are positive pairs. That is, random state initializations for the same task μ should produce similar trajectory embeddings; we term this positive sampling [17].

6.2 Hidden state augmentations

While random starting states on the grid environments should be sufficient augmentations, one could also use different hidden state initializations of the exploration DDQN, whereby per the hidden state hypothesis, at each time step in a trajectory, the hidden state learns more information about the task at hand. That is, the hidden states of the recurrent exploration policy learns a history, i.e. buffer, of information $\boldsymbol{h}_t \in \mathbb{R}^k$ about the task at each time step t. We let $H_{\text{buff}} = [\boldsymbol{h}_1 \cdots \boldsymbol{h}_H]$ be a buffer of this information. Thus, if we want to create positive augmentations of the trajectory τ^{exp} in task setting μ, we can sample a hidden state $\boldsymbol{h}_i \sim H_{\text{buff}}$ from the buffer and pass it in as the initial hidden state of the recurrent exploration policy $\boldsymbol{h}'_0 := \boldsymbol{h}_i$. Note, the last element of the buffer does not necessarily contain the most information about the task, i.e. sometimes during exploration, the agent goes to the map–which will give us maximal information about the task–and then it might wander around for a while after so the final state does not necessarily correspond to being on the map. Thus, we do not always expect $\boldsymbol{h}_H$ to contain maximal encoded information about the task. However, we observe that as the exploration policy is trained, the agent learns to end the episode after encountering the map, so the expectation of $\boldsymbol{h}_H$ containing maximally useful information increases over time.

6.3 Negative Sampling Augmentations

On the other hand, we generate K "negative" examples as follows: in a different task $\mu_i \neq \mu$ and from a random starting state s_i^-, roll out the trajectory $(\tau_i^{\text{exp}})^- = (s_0^-, a_0^-, r_0^-, s_1^-, \dots) \sim \pi_\phi^{\text{exp}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ with $\boldsymbol{z} \sim F(\boldsymbol{z}|\mu_i)$, for $i = 1, \dots, K$. Then we embed these trajectories as $\boldsymbol{\alpha}_i^- = g_\omega((\tau_i^{\text{exp}})^-)$. We want to force $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}_i^-$ to be far apart in the trajectory embedding space because we hypothesize that exploitation can more easily infer the task encoding $\boldsymbol{z} \sim g_\omega(\boldsymbol{z}|\tau^{\text{exp}})$ when the trajectory embeddings for distinct tasks are clustered apart and the trajectory embeddings for similar tasks are clustered together. In fact, this hypothesis is based on our observation of the t-SNE plots of trajectory embeddings as we perform meta-training, as shown in the accompanying t-SNE figure.

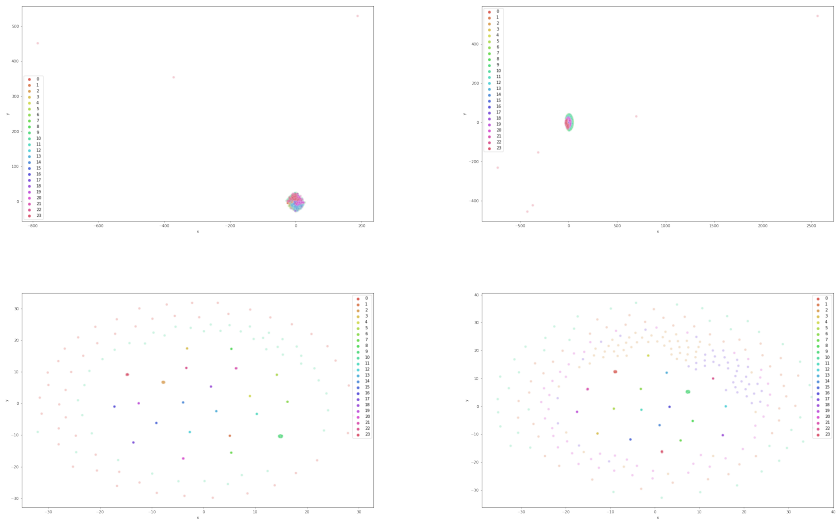

Figure 5: Trajectory embeddings after 2000 rollouts (top left); Final hidden state embeddings after 2000 rollouts (top right); Trajectory embeddings after 10000 rollouts (bottom left); Final hidden state embeddings after 10000 rollouts (bottom right)

6.4 Contrastive Learning

We let

$$X := \begin{bmatrix} \boldsymbol{\alpha}^+ & \boldsymbol{\alpha}_1^- & \cdots & \boldsymbol{\alpha}_K^- \end{bmatrix}^\top \in \mathbb{R}^{(K+1)\times d} \tag{13}$$

where $\boldsymbol{\alpha}^+, \boldsymbol{\alpha}_i^- \in \mathbb{R}^d$ for $i = 1, \ldots, K$. Then, as in SimCLR [17], we form the similarity vector:

$$\boldsymbol{s} = \begin{bmatrix} \text{sim}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^+) \\ \text{sim}(\boldsymbol{\alpha}, \boldsymbol{\alpha}_1^-) \\ \vdots \\ \text{sim}(\boldsymbol{\alpha}, \boldsymbol{\alpha}_K^-)) \end{bmatrix} := \begin{bmatrix} (\boldsymbol{\alpha}^\top \boldsymbol{\alpha}^+)/(||\boldsymbol{\alpha}||||\boldsymbol{\alpha}^+||) \\ (\boldsymbol{\alpha}^\top \boldsymbol{\alpha}_1^-)/(||\boldsymbol{\alpha}||||\boldsymbol{\alpha}_1^-||) \\ \vdots \\ (\boldsymbol{\alpha}^\top \boldsymbol{\alpha}_K^-)/(||\boldsymbol{\alpha}||||\boldsymbol{\alpha}_K^-||) \end{bmatrix} \tag{14}$$

Then the loss for the positive pair $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^+$ is computed as the negative log-likelihood:

$$\mathcal{L}_{\text{simclr}} = -\log \frac{\exp(\text{sim}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^+)/T)}{\sum_{i=1}^K \exp(\text{sim}(\boldsymbol{\alpha}, \boldsymbol{\alpha}_i^-)/T)} \tag{15}$$

where we temper the softmax distribution with temperature T.

6.5 Results & Analysis

Figure 6: Contrastive learning (blue) compared against DREAM baseline (orange) on basic map grid environment.

We observe, as seen in Figure 4, that contrastive learning facilitates exploration. In fact, exploration reward quickly attains optimal rewards in less than 2k exploration episodes, compared to the baseline of over 30k exploration episodes. We interpret exploration reward as maximizing the mutual information between the exploration trajectory embedding τ^{exp} and the task encoding $\boldsymbol{z}$; therefore, we quickly learn to maximize the similarity between these two. However, it should be noted that maximizing the mutual information does not necessarily imply that the task encoding $\boldsymbol{z} \sim F(\boldsymbol{z}|\mu)$ is a good or correct representation of the task. As such, we ran an experiment in which we delayed using the contrastive loss on the trajectory embedder so that we can learn a good encoder $F(\boldsymbol{z}|\mu)$ before forcing mutuality of $\boldsymbol{z}$ and τ^{exp}, i.e. $I(\tau^{\text{exp}}; \boldsymbol{z})$. Nonetheless, as shown in Figure X, we found that delaying contrastive training did not help exploitation rewards converge more quickly.

6.6 Generalized Algorithm

We enumerate a generalization of our contrastive loss algorithm as follows:

1. The exploration policy is rolled out on task μ to generate trajectory τ^{μ}.
2. τ^{μ} is passed to the trajectory embedder to get embedding $\boldsymbol{\alpha}$.
3. The exploration policy is then rolled out on task μ but with different starting positions (and/or initial hidden states) to get embeddings $\boldsymbol{\alpha}_i$ for i ranging from 1 to k.
4. Compute the consistency component of the contrastive loss through $\sum_i \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\alpha}_i)$ where $\mathcal{L}$ is a distance metric
5. Roll out the exploration policy n times on different tasks to get representations $\boldsymbol{\beta}_i$ for i ranging from 1 to n.
6. Compute the negative sampling part of the contrastive loss via $-\sum_i \mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\beta}_i)$ and return the contrastive loss as the sum of the components.

7 Semi-Supervised Meta-Reinforcement Learning

Literature, such as [18] and [19], has applied meta-learning on tasks constructed from an unlabeled dataset. Instead, we focus on the setting where a small number of tasks are known and we must meta-learn the other tasks in a semi-supervised way.

7.1 Problem Setting

In particular, we consider the grid environment for which only a subset of tasks, i.e. the configuration of bus destinations, $\{\mu_{i_1}, \ldots, \mu_{i_n}\} \subset \{\mu_1, \ldots \mu_K\}$ where $1 < i_m < K$ for $m = 1, \ldots, n$ is known.

Thus, we denote the known problem IDs as $\mu_{i_1}, \ldots, \mu_{i_n}$ or, for simplicity, and the unobserved problem IDs as $\zeta_1, \ldots, \zeta_{K-n}$. In the four bus environment, there are $K = 4!$ tasks and we select a subset of $n = 8$ to be supervised.

7.2 Contrastive Learning

Contrastive learning, as described in the previous section, is naturally amenable to semi-supervised learning settings. That is, we begin by performing normal supervised DREAM on the supervised tasks $\mu_{i_1}, \ldots, \mu_{i_n}$ for some number of time steps, preferably until the n supervised tasks are solved, essentially pre-training the exploration and exploitation policies on the supervised tasks. In the case of the four bus environment, this tasks roughly 32k episodes.

Once we have pre-trained the policies, we continue training on *both* the supervised and unsupervised problem IDs. Recall, we train the exploitation policy conditioned on $\boldsymbol{z} = q_\omega(\tau^{\text{exp}})$ if the meta-training trial is odd-numbered and conditioned on $\boldsymbol{z} \sim \mathcal{N}(F_\Psi(\mu), \rho^2 \boldsymbol{I})$ if the meta-training trial is even-numbered.

Therefore, during meta-training, on even trials, we need μ to pass into the encoder F_Ψ. Hence, we only sample a task from the supervised tasks, i.e. one of the $\mu_{i_1}, \ldots, \mu_{i_n}$. Then we proceed to normally train the exploration policy, which should eventually find the map and, hence, μ. Thus, we train the exploitation policy, encoder, and decoder as in regular DREAM.

However, on odd trials, since we use the trajectory decoder q_ω, we sample a task from all tasks, i.e. both supervised and unsupervised tasks from which we get $\boldsymbol{z} = q_\omega(\tau^{\text{exp}})$ and proceed to update the exploration and exploitation policy as usual. We also train the trajectory embedder using contrastive learning by pushing similar embeddings close together and dissimilar, i.e. trajectories for different tasks, far apart.

7.3 Results & Analysis

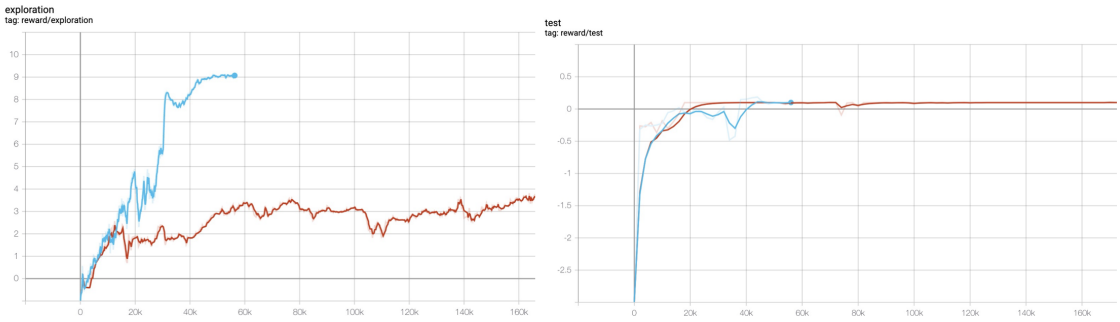

Figure 7: Semi-supervised learning with contrastive loss (blue), after 32k steps of pre-training with 8 tasks, reaches optimal exploration rewards, i.e. maximizes mutual information between the task encoding and the trajectory embedding, at a much faster rate than the baseline semi-supervised learning without contrastive loss (red).

We pre-train for 32k steps on the 8 supervised tasks with regular DREAM. While it is not reflected in the test exploitation reward graph since we average the rewards for all 24 tasks, we obtain optimal exploitation rewards for the 8 known tasks. Thus, we solve the 8 tasks before we continue with semi-supervised training. When we continue to train after 32k episodes, the exploitation reward falls and the solutions for all tasks involve simply walking to the destination and avoiding all the buses. However, we see a decreasing contrastive loss across all tasks, which suggests that the trajectory embedder is effectively able to cluster trajectories by task. This indicates that the exploitation policy recieves latent information that successfully identifies the appropriate task and is still unable to effectively leverage that information. Ultimately, this is likely due to the fact that the exploitation policy explicitly leverages latent information in the form of the encoded task-id and the trajectory embedder for the supervised tasks

7.4 Future Work: Semi-supervised k-means

Predicated on our hypothesis that trajectories for the same task cluster in the embedding space, we posit that we can infer the problem ID from the clustered trajectories. In particular, trajectories rolled out on the same supervised task sampled from $\mu_{i_1}, \ldots, \mu_{i_n}$ should all cluster around one of the appropriate n centroids with assignment label corresponding to the problem ID.

7.5 Training Process

We begin the process by collecting $M \sim 10^3$ supervised and unsupervised trajectories on problem IDs $\mu_1, \ldots, \mu_K$.

For the first meta-training trial, we initialize the centroids and label assignments as follows. For the supervised trajectories, we set the n centroids to be the mean of the trajectories that have labels given by the supervised problem IDs. The centroids are given labels corresponding to the labels of the clustered trajectories.

For the unsupervised trajectories, we randomly initialize $K - n$ other centroids and assign the unsupervised trajectories to the closest centroid.

Now, every $N \sim 10^3$ meta-trials, we perform a k-means update to change cluster centroids and assignments. However, note, the label assignments for supervised trajectories do not ever change. We wait for N trials before updating to induce stability in the learned encoder for the problem IDs. Furthermore, we never re-initialize the centroids or label assignments to ensure stability, i.e. that the centroids and assignments do not differ too much, every time we update them every N trials.

We update the encoder $F_\Psi(\boldsymbol{z}|\mu)$, the exploitation policy $\pi_\theta^{\text{task}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$, and the decoder $g_\omega(\tau^{\text{exp}})$ every meta-training trial, after warm-up period. If we are on an odd-numbered meta-trial, we use the trajectory decoder $\boldsymbol{z} = g_\omega(\tau^{\text{exp}})$ to sample the latent task encoding. If we are on an even-numbered trial, then we use the encoder $\boldsymbol{z} \sim F_\Psi(\boldsymbol{z}|\mu)$; however, in cases where the trajectory is unsupervised, i.e. for the unsupervised tasks $\zeta_1, \ldots, \zeta_{K-n}$, we infer the problem ID from the k-means centroids.

To infer the problem ID, we first roll out the trajectory and compute its embedding $\boldsymbol{z} = g_\omega(\tau^{\text{exp}})$. We find the centroid closest to it in squared Euclidean distance. Then the inferred label $\hat{\mu}$ is the label of the closest centroid. Hence, we can now update the encoder, decoder, and exploitation policy as in normal DREAM.

7.6 Using MAML to Meta-learn and Stabilize Task Encoder

We update the encoder $F_\Psi(\boldsymbol{z}|\mu)$ every trial and update the k-means centroids and assignments every N trials. Then any time the unsupervised problem IDs μ change, label asignment when we update, the encoder is no longer valid for those problem IDs, which leads to instability. While eschewing re-initialization of centroids and assignments every N trials creates stability, this is not a strong guarantee. We propose an alternative method that leverages MAML [7] to meta-learn a task encoder $F_\Psi(\boldsymbol{z}|\mu)$ that is robust to changes in problem ID assignments for unsupervised tasks. In particular, we learn meta-parameters Ψ for the task encoder F_Ψ across various permutations of the unsupervised problem IDs $\tau_i^{\text{train}} = \text{perm}(\{\zeta_1, \ldots, _{K-n}\})$, where each permutation represents a different meta-training "task". Then at meta-test time when we are using a particular permutation of the unsupervised IDs to train the k-means model, we fine-tune the model with a few SGD steps to rapidly adapt to the particular permutation $\tau^{\text{test}} = \{\zeta_{i_1}, \ldots, \zeta_{i_{K-n}}\}$ where $1 \leq i_m \leq K - n$ for $m = 1, \ldots, K - n$.

Algorithm 4 Semi-supervised DREAM meta-training trial t

1. Sample problem $\mu \sim p(\mu)$
2. Semi-supervised k-means
Only initialize centroids and label assignments at the beginning of meta-training for stability.
if meta-trial $t = 0$ **then**:
  $A_{\text{sup}} = \{\}$
  $A_{\text{unsup}} = \{\}$ ▷ Collect supervised and unsupervised trajectories
  for $M \sim 500$ iterations **do**:
    Sample task ID $\mu \sim \{\mu_{i_1}, \ldots, \mu_{i_n}, \zeta_1, \ldots, \zeta_{K-n}\}$
    Rollout $\tau^{\text{exp}} \sim \pi_\theta^{\text{exp}}(\boldsymbol{a}|\boldsymbol{s}, \tau_{:t}^{\text{exp}})$
    $\boldsymbol{\alpha} = g(\tau^{\text{exp}})$
    if $\boldsymbol{\alpha}$ has supervised task ID **then**:
      $A_{\text{sup}} \leftarrow A_{\text{sup}} \cup \alpha$
    else
      $A_{\text{unsup}} \leftarrow A_{\text{unsup}} \cup \alpha$
    end if
  end for
  Set $A \leftarrow A_{\text{sup}} \cup A_{\text{unsup}}$
  ▷ Initialize label assignments for supervised trajectories
  for $\alpha^\mu \in A_{\text{sup}}$ with problem ID μ **do**:
    Set $c^{(\boldsymbol{\alpha}^\mu)} := \mu$ ▷ We know the labels for supervised trajectories
  end for
  ▷ Initialize centroids for supervised trajectories
  for $j = 1, \ldots, n$ **do**:

$$\boldsymbol{m}_j^{\text{sup}} = \frac{\sum_{\boldsymbol{\alpha}^\mu \in A_{\text{sup}}} \mathbb{1}\{c^{(\boldsymbol{\alpha}^\mu)} = j\}\boldsymbol{\alpha}^\mu}{\sum_{\boldsymbol{\alpha}^\mu \in A_{\text{sup}}} \mathbb{1}\{c^{(\boldsymbol{\alpha}^\mu)} = j\}} \in \mathbb{R}^d$$

  end for
  Randomly initialize unsupervised cluster centroids $\boldsymbol{m}_1^{\text{unsup}}, \ldots, \boldsymbol{m}_{K-n}^{\text{unsup}} \in \mathbb{R}^d$
end if
# k-means update, only every $N \sim 10^3$ trials:
if $t \equiv 0 \mod N$ **then**:
  while not converged **do**:
    for $\boldsymbol{\alpha} \in A_{\text{unsup}}$ **do**:

$$c^{(\boldsymbol{\alpha})} = \arg\min_j ||\boldsymbol{\alpha} - \boldsymbol{m}_j||_2^2$$

      ▷ Assign unsupervised trajectories to clusters (we already have the task ids for supervised)
    end for
    for $j = 1, \ldots, K$ **do**:

$$\boldsymbol{m}_j = \frac{\sum_{\boldsymbol{\alpha} \in A} \mathbb{1}\{c^{(\boldsymbol{\alpha})} = j\}\boldsymbol{\alpha}}{\sum_{\boldsymbol{\alpha} \in A} \mathbb{1}\{c^{(\boldsymbol{\alpha})} = j\}}$$

      ▷ Update all (supervised and unsupervised) cluster centroids
    end for
  end while
end if
3. Compute task embedding (supervised and unsupervised)
if $t \equiv 0 \mod 2$ **then**
  # If task unsupervised (not observable), estimate $\hat{\mu}$ from clusters
  if $\mu \in \{\zeta_1, \ldots, \zeta_{K-n}\}$ **then**:
    Decode $\boldsymbol{\alpha} = g_\omega(\tau^{\text{exp}})$
    # Infer problem ID from k-means
    Set $\hat{\mu} = \arg\min_j ||\boldsymbol{\alpha} - \boldsymbol{m}_j||_2^2$
    Compute problem encoding $\boldsymbol{z} \sim F_\Psi(\boldsymbol{z}|\hat{\mu})$
  else
    # If task is supervised, directly encode
    Compute problem encoding $\boldsymbol{z} \sim F_\Psi(\boldsymbol{z}|\mu)$
  end if
else
  Compute embedding $\boldsymbol{z} = g_\omega(\tau^{\text{exp}})$
end if
4. Exploration episode
Roll out exploration policy $\tau^{\text{exp}} \sim \pi_\phi^{\text{exp}}(\boldsymbol{a}_t|\boldsymbol{s}_t, \tau_{:t}^{\text{exp}})$
Update π_ϕ^{exp} and q_ω to maximize $I(\tau^{\text{exp}}; \boldsymbol{z})$
5. Exploitation episode
Every other episode, choose $\boldsymbol{z} \sim q_\omega(\boldsymbol{z}|\tau^{\text{exp}})$
Roll out exploitation policy $\pi_\theta^{\text{task}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$
Update π_θ^{task} and F_Ψ

8 Conclusion and Future Work

Our goal was to put DREAM under a number of limitations and develop ways to make it more efficient and able to perform under them. The authors felt that DREAM was substantially better at a variety of hard exploration tasks than originally thought and remark at how certain modifications - while appearing relatively minor - could cause great training instability and a lack of convergence. One of our main takeaways was that our PEARL-based method, when trained without a hidden state, was around as sample efficient and achieved optimal rewards on the different environments like vanilla DREAM. The nature of the algorithm's context embedding is likely to only contain information relative to task identity - something that might be much more usable than final cell state for an exploration trajectory embedding - which leads to a future avenue of work. Our other main takeaway was that the inclusion of our contrastive loss was relatively useful for tasks that did not have a trained ID embedding. As decoupled meta-RL frameworks become more standard, we hope that our research can provide insight into how a state-of-the-art algorithm like DREAM works and what modifications can help improve performance.

9 Contributions

The nature of the project changed quite a bit relative to the original project proposal as we wanted to focus on trying a number of modifications to DREAM rather than devote additional type to tuning and debugging simulator issues for the Stanford Pupper virtual environment. Our contributions are split as follows:

1. Sergio Charles: Worked on the theory and development of the PEARL-based task specific exploration algorithm, theory and development of contrastive loss / semi-supervised experiments, and wrote sections 5, 6, and 7 of the paper.
2. Raghav Samavedam: Worked on the theory and initial development of the contrastive loss / semi-supervised approach, experimentation of cell state based trajectory embeddings, and wrote sections 1, 2, and 3 of the paper.
3. Arnav Joshi: Worked on development of contrastive loss and theory of k-means based semi-supervised method, engineering didactic environments and t-SNE visualizations, and wrote sections 4, 8, and 9 of the paper.

References

[1] Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices, 2021.

[2] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables, 2019.

[3] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners, 2020.

[4] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago, 2018.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[8] Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning, 2019.

[9] Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search, 2020.

[10] Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients, 2018.

[11] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Jie Tan, and Chelsea Finn. Norml: No-reward meta learning, 2019.

[12] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards, 2019.

[13] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies, 2019.

[14] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A. Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference, 2019.

[15] Jin Zhang, Jianhao Wang, Hao Hu, Tong Chen, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. Metacure: Meta reinforcement learning with empowerment-driven exploration, 2021.

[16] Swaminathan Gurumurthy, Sumit Kumar, and Katia Sycara. Mame : Model-agnostic meta-exploration, 2019.

[17] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[18] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning, 2019.

[19] Siavash Khodadadeh, Sharare Zehtabian, Saeed Vahidian, Weijia Wang, Bill Lin, and Ladislau Bölöni. Unsupervised meta-learning through latent-space interpolation in generative models, 2020.