Appendix A. Additional Related Work

This section extends the discussion on the related works from Section 2.

Some prior works train a **single fixed dynamics model** but introduce additional constraints that ensure the latent dynamics are locally linear (Watter et al., 2015; Banijamali et al., 2018) or that the learned dynamics model is invariant to translation dynamics (Fragkiadaki et al., 2015). However, such approaches fail when the parameters of the underlying dynamics model change. In theory, one could learn a new *expert* dynamics model per task, but that is computationally expensive and requires access to the unseen environments. Xian et al. (2021) generates the weights of an *expert* dynamics model by using the environment context as an input. Specifically, they proposed an algorithm called HyperDynamics where they use a HyperNetwork (Ha et al., 2016; Chang et al., 2019; Klocek et al., 2019; Meyerson and Miikkulainen, 2019) to generate the weights of the dynamics model.Similar to our work, Lee et al. (2020) also introduced additional loss terms when training the agent. However, their objective is to encourage the context encoding to be useful for predicting both forward (next state) and backward (previous state) dynamics while being temporally consistent. Asadi et al. (2018) present results when bounding the model class to Lipschitz functions but assume that the given MDP is Lipschitz. We instead show that this constraint can be placed on the representation learning objective for better results in the continual learning setting, even when the original BC-MDP is not Lipschitz.

Our work is related to the broad area of **multitask RL and transfer learning** (Caruana, 1997; Zhang et al., 2014; Kokkinos, 2017; Radford et al., 2019; Rajeswaran et al., 2016). Previous works have looked at multi-task and transfer learning from the perspective of policy transfer (Rusu et al., 2015; Yin and Pan, 2017), policy reuse (Fernández and Veloso, 2006; Barreto et al., 2016), representation transfer (Rusu et al., 2016; Devin et al., 2017; Andreas et al., 2017; Sodhani et al., 2021b) etc. One unifying theme in these works is that the agent is finetuned on the new environment while we focus on setup where there are no gradient updates when evaluating on the unseen environment. Zhang et al. (2021b) also uses task metrics to learn a context space, but focus on the rich observation version of the Hidden-Parameter MDP (HiP-MDP) setting (Doshi-Velez and Konidaris, 2013) with access to task ids. Similarly, Perez et al. (2020) also treats the multi-task setting as a HiP-MDP by explicitly designing latent variable models to model the latent parameters, but require knowledge of the structure upfront, whereas our approach does not make any such assumptions.

Several works proposed a *mixture-of-experts* based approach to adaptation where an expert model is learned for each task (Doya et al., 2002; Neumann et al., 2009; Peng et al., 2016) or experts are shared across tasks (Igl et al., 2020; Sodhani et al., 2021b). Teh et al. (2017) additionally proposed to distill these experts in a single model that should work well across multiple tasks. While these systems perform reasonably well on the training tasks, they do not generalize well to unseen tasks.

Appendix B. Implementation Details

We provide additional details of the main algorithm, environment setup, compute used, baselines, and hyperparameter information.

B.1. Algorithm

We provide pseudocode for the main algorithm in Algorithm 1 and the context loss update in Algorithm 2.

Algorithm 1 Training ZeUS algorithm.

Require: Actor, Critic, Dynamics Model T, Reward Model R, Observation Encoder ϕ, Context Encoder ψ, Replay Buffer $\mathcal{D}$.

for each timestep $t = 1..T$ **do**
  for each $\mathcal{E}_i$ **do**
    $a_t^i \sim \pi^i(\cdot|s_t^i)$
    $s_t'^i \sim p^i(\cdot|s_t^i, a_t^i)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t^i, a_t^i, r(s_t^i, a_t^i), s_t'^i)$
    UPDATE_CRITIC($\mathcal{D}$)
    UPDATE_ACTOR($\mathcal{D}$)
    UPDATE_USING_CONTEXT_LOSS($\mathcal{D}$)
  end for
end for

Algorithm 2 Update Using Context Loss

Require: Batches of data sampled from the Replay Buffer $\mathcal{D}$, Dynamics Model T, Reward Model R, Observation Encoder ϕ, Context Encoder ψ, Learning rates α_ψ, α_T and α_R.

for each batch **do**
  Compute $\mathcal{L}(\phi, \psi, T, R))$ using Equation (2)
  $\psi \leftarrow \psi - \alpha_\psi \nabla_\theta \sum_i \mathcal{L}$
  $T \leftarrow T - \alpha_T \nabla_\theta \sum_i \mathcal{L}$
  $R \leftarrow R - \alpha_R \nabla_\theta \sum_i \mathcal{L}$
end for

B.2. Setup

Environments For tasks with varying dynamics, we use the Finger, Cheetah and Walker environments. These are Mujoco (Todorov et al., 2012)[7] based tasks that are available as part of the DMControl Suite Tassa et al. (2018)[8]. We use MTEnv Sodhani et al. (2021a)[9] to interface with the environments. In the Finger Spin task, the agent has to manipulate (continually rotate) a planar *finger*. We vary the size across different tasks and the specific values (for each mode) are listed in Table 2. In the Cheetah Run task, a planar bipedal cheetah has to run as fast as possible (up to a maximum velocity). In the Walker Walk tasks, a planar walker has to learn to walk. We consider two cases here: (i) varying the friction coefficient between the walker and the ground, and (ii) the length of the foot of the walker.

For tasks with varying reward function, we use the Cheetah and the Sawyer Peg environments from Zhao et al. (2020)[10]. We highlight some challenges in the evaluation: In the cheetah environment, the reward depends on the difference in the magnitude of the agent's velocity and the target velocity. In each run of the algorithm, the target velocities are randomly sampled. We noticed that the cheetah can easily match the target velocity when the target velocity is small and makes a larger error when the target velocity is large. In practice, the performance of the cheetah (as measured in terms of

7. https://www.roboti.us
8. https://github.com/deepmind/dm_control
9. https://github.com/facebookresearch/mtenv/commit/7fdec15f7e842bce4c17f4f3328d9d6fdc79d7fc
10. https://github.com/tonyzhaozh/meld

episodic rewards) can vary a lot depending on the sampled target velocities. To make the comparison fair across the different baselines, we fix the values of the target velocities (for train and for eval) instead of sampling them. In the Peg-Insertion task, the reward has an extra "offset" term as shown in the implementation but this does not match the description in Zhao et al. (2020). We use the version of reward function without the offset, as described in Zhao et al. (2020).

All the algorithms are implemented using PyTorch (Paszke et al., 2017)[11]. We use the MTRL codebase[12] as the starting codebase.

Compute Resources Unless specified otherwise, all the experiments are run using Volta GPUs with 16 GB memory. Each experiment uses 1 GPU and 10 CPUs (to parallelize the execution in the environments). The experiments with HyperDynamics model are run with Volta GPUs with 32 GB Memory. For the Cheetah-Run-v0, Finger-Spin-v0, Walker-Walk-v0 and Walker-Walk-v1, training CADM and ZeUS takes about 44 hours, training HyperDynamics takes about 62 hours and training UP-OSI takes 24 hours (all for 1.2 M steps). For Cheetah-Run-v1 and Sawyer-Peg-v1, training CADM and ZeUS takes about 25 hours while HyperDynamics takes about 32.5 hours (all for 300K steps). These times include the time for training and evaluation.

In Table 1, we show that a two-layer feedforward network can be trained to infer the context value from last 5 observation-action tuples. The setup is modeled as a regression problem where the observation-action transition tuples are obtained from a random policy.

Table 1: Loss when training the model to infer the context

Environment Name	Loss
Cheetah-Run-v0	5.12×10^{-5}
Finger-Spin-v0	7.45×10^{-5}
Walker-Walk-v0	7.22×10^{-5}
Walker-Walk-v1	6.41×10^{-5}

B.2.1. BASELINES

We provide additional implementation related details for the baselines. For a summary of the baselines, refer Section 6.2.

1. *Context-aware Dynamics Model (CaDM)*: Lee et al. (2020) proposed a two stage pipeline: (a) use a context encoder to obtain a context vector given the recent interactions and (ii) perform online adaption by conditioning the forward dynamics model on the context vector. CaDM is shown to outperform vanilla dynamics models which do not use the interaction history, stacked dynamics models which use the interaction history as an input, and Gradient and Recurrence-based meta learning approaches (Nagabandi et al., 2018).

2. *UP-OSI*: Yu et al. (2017) proposed learning two components: (i) Universal Policy (UP) and (ii) On-line System Identification model (OSI). The universal policy is trained over a wide range of dynamics parameters so that it can operate in a previously unseen environment (given

11. https://pytorch.org/
12. https://github.com/facebookresearch/mtrl/commit/eea3c99cc116e0fadc41815d0e7823349fcc0bf4

access to the parameters of the dynamics model). It is modeled as a function of the dynamic parameters μ, such that $a_t = \pi_\mu(s_t)$ and is trained offline, in simulation, without requiring access to the real-world samples. The goal of the universal policy is to generalize to the space of the dynamics models. The on-line system identification model is trained to identify the parameters of the dynamics model conditioned on the last k state-action transition tuples i.e. $\mu = \phi((s_i, a_i, s'_i, r_i), \forall i \in \{1, ..., k\})$, where ϕ is the OSI module. ϕ is trained using a supervised learning loss by assuming access to true parameters of the dynamics model. During evaluation (in an unseen environment), the OSI module predicts the dynamic parameters at every timestep. The universal policy uses these inferred dynamics parameters to predicts the next action.The system identification module is trained to identify a model that is good enough to predict a policy's value function that is similar to the current optimal policy. Following the suggestion by (Yu et al., 2017), we initially train the OSI using the data collected by UP (when using true model parameters) and then switch to the data collected using inferred parameters. Specifically, the paper suggested using first 3-5 iterations (out of 500 iterations) for training with the ground truth parameters. We scaled the number of these warmup iterations to match the number of updates in our implementation and report the results using the same. During evaluation, we report the performance of UP using the inferred parameters (UP-*inferred*), as recommended in the paper.

3. *HyperDynamics*: Xian et al. (2021) proposed learning three components: (i) an encoding module that encodes the agent-environment interactions, (ii) a hypernetwork that generates the weight for a dynamics model at the current timestep, and (iii) a (target) dynamics model that uses the weights generated by the hypernetwork. All the components (and the policy) are trained jointly in an end-to-end manner. HyperDynamics is shown to ourperform both ensemble of experts and meta-learning based approaches (Nagabandi et al., 2018).

4. *Context-conditioned Policy*: We train a context encoder that encoders the interaction history into latent context that is passed to the policy and the dynamics. This approach can be thought of as an ablation of the ZeUS algorithm with $\alpha_\psi = 0$, or without the context learning error (from Equation (2)). We label this baseline as *Zeus-no-context-loss*.

We note that when training on the environments with varying reward dynamics, we concatenate the reward along with the environment observation (as done in Zhao et al. (2020)).

B.3. Environment Details

Table 2: Parameter values for different modes for the Finger environments (Finger-Spin-v0) when varying the size of the finger across the tasks.

Mode	Values
Train / Eval	[0.05, 0.0625, 0.075, 0.0875, 0.15, 0.1625, 0.175, 0.1875]
Eval Interpolation	[0.1, 0.1125, 0.125, 0.1375]
Eval Extrapolation	[0.01, 0.0125, 0.025, 0.0375, 0.2, 0.2125, 0.35, 0.5]

Table 3: Parameter values for different modes for the Cheetah environments (Cheetah-Run-v0) when varying the length of cheetah's torso across the tasks.

Mode	Values
Train / Eval	[0.6, 0.65, 0.7, 0.75, 1, 1.05, 1.1, 1.15, 1.25, 1.4, 1.5]
Eval Interpolation	[0.8, 0.85, 0.9, 0.95]
Eval Extrapolation	[0.3, 0.4, 0.5, 0.55, 1.2, 1.25, 1.4, 1.5]

Table 4: Parameter values for different modes for the Walker environments (Walker-Walk-v0) when varying the friction coefficient between the walker and the ground.

Mode	Values
Train / Eval	[0.8, 0.85, 0.9, 0.95, 1.2, 1.25, 1.3, 1.35]
Eval Interpolation	[1.0, 1.05, 1.1, 1.15]
Eval Extrapolation	[0.3, 0.5, 0.7, 0.75, 1.4, 1.45, 1.7, 1.9]

Table 5: Parameter values for different modes for the Walker environments (Walker-Walk-v1) when varying the length of walker's foot.

Mode	Values
Train / Eval	[0.09, 0.095, 0.1, 0.105, 0.13, 0.135, 0.14, 0.145]
Eval Interpolation	[0.11, 0.115, 0.12, 0.125]
Eval Extrapolation	[0.03, 0.06, 0.08, 0.085, 0.15, 0.155, 0.18, 0.21]

Table 6: Values for the target velocity for the Cheetah environments (Cheetah-Run-v1).

Mode	Values
Train	[0., 0.43, 0.86, 1.29, 1.71, 2.14, 2.57, 3.0]
Eval	[0.4, 0.78, 1.11, 1.47, 1.83, 2.18, 2.55, 2.9]

Table 7: Range for sampling the values for the Sawyer-Peg environments (Sawyer-Peg-v0)

Mode	Values
x_range_1	(0.44, 0.45)
x_range_2	(0.6, 0.61)
y_range_1	(-0.08, -0.07)
y_range_2	(0.07, 0.08)

B.4. License

1. Mujoco: Commercial (with Trial Version) https://www.roboti.us/license.html
2. DeepMind Suite: Apache https://github.com/deepmind/dm_control/blob/master/LICENSE
3. MTEnv: MIT License https://github.com/facebookresearch/mtenv/blob/main/LICENSE
4. Meld Codebase: https://github.com/tonyzhaozh/meld
5. PyTorch: https://github.com/pytorch/pytorch/blob/master/LICENSE
6. MTRL: MIT License https://github.com/facebookresearch/mtrl/blob/main/LICENSE
7. Hydra: MIT License https://github.com/facebookresearch/hydra/blob/master/LICENSE

B.5. Hyperparameter Details

In this section, we provide hyper-parameter values for each of the methods in our experimental evaluation. We also describe the search space for each hyperparameter. In Table 9 and Table 8 , we provide the hyperparameter values that are common across all the methods.

Figure 6: Variation in `Cheetah-Run-v0` tasks.

Table 10: Hyperparameter values for Context Aware Dynamics Model

Hyperparameter	Hyperparameter values	Environment
β	0.5	Cheetah-Run-v0
β	0.5	Finger-Spin-v0
β	0.5	Walker-Walk-v0
β	0.5	Walker-Walk-v1
β	0.5	Cheetah-Run-v1
β	0.5	Sawyer-Peg-v0

Appendix C. Additional Results

We present additional results not in the main paper.

C.1. Handling nonstationarity in the training environments

In Figure 7 we show performance on ZeUS and baselines on the training environments.

Table 8: Hyperparameter values that are common across all the methods for Cheetah-Run-v0, Finger-Spin-v0, Walker-Walk-v0 and Walker-Walk-v1 (envs with varying dynamics)

Hyperparameter	Hyperparameter values
batch size (per task)	128
network architecture	feedforward network
non-linearity	ReLU
policy initialization	standard Gaussian
exploration parameters	run a uniform exploration policy 1500 steps
# of samples / # of train steps per iteration	1 env step / 1 training step
policy learning rate	3e-4
Q function learning rate	3e-4
Critic update frequency	2
optimizer	Adam
beta for Adam optimizer for policy	(0.9, 0.999)
Q function learning rate	3e-4
beta for Adam optimizer for Q function	(0.9, 0.999)
Discount	.99
Episode length (horizon)	500
Reward scale	1.0
actor update frequency	2
actor log stddev bounds	$[-10, 2]$
number of layers in actor/critic	2
actor/critic hidden dimension	1024
number layers in dynamics model	1
dynamics hidden dimension	512
number of encoder layers	4
number of filters in encoder	32
Replay buffer capacity	400000
Temperature Adam's β_1	0.9
Init temperature	0.1
Context Length	5

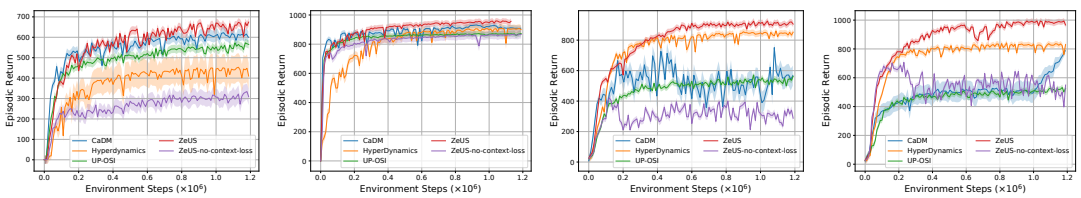

Figure 7: We compare the performance of the proposed ZeUS algorithm with *CaDM*, *UP-OSI*, *HyperDynamics* and *ZeUS-no-context-loss* algorithms on the training environments for four families of tasks with different dynamics parameters.

Table 9: Hyperparameter values that are common across all the methods for Cheetah-Run-v1 and Sawyer-Peg-v0 (envs with varying reward functions)

Hyperparameter	Hyperparameter values
batch size (per task)	128
network architecture	feedforward network
non-linearity	ReLU
policy initialization	standard Gaussian
exploration parameters	run a uniform exploration policy 10000 steps
# of samples / # of train steps per iteration	1 env step / 1 training step
policy learning rate	3e-4
Q function learning rate	3e-4
Critic update frequency	2
optimizer	Adam
beta for Adam optimizer for policy	(0.5, 0.999)
Q function learning rate	3e-4
beta for Adam optimizer for Q function	(0.5, 0.999)
Discount	.99
Episode length (horizon)	40
Reward scale	1.0
actor update frequency	2
actor log stddev bounds	$[-10, 2]$
number of layers in actor/critic	2
actor/critic hidden dimension	1024
number layers in dynamics model	1
dynamics hidden dimension	512
number of encoder layers	4
number of filters in encoder	32
Replay buffer capacity	400000
Temperature Adam's β_1	0.5
Init temperature	0.1
Context Length	5

C.2. Handling nonstationarity in the Interpolation environments

In Figure 8 we show performance on ZeUS and baselines on evaluation environments that are interpolated from the train environments.

Table 11: Hyperparameter values for ZeUS

Hyperparameter	Hyperparameter values	Environment
α_ψ	1.0	Cheetah-Run-v0
α_ψ	0.5	Finger-Spin-v0
α_ψ	2.0	Walker-Walk-v0
α_ψ	0.1	Walker-Walk-v1
α_ψ	1.0	Cheetah-Run-v1
α_ψ	1.0	Sawyer-Peg-v0

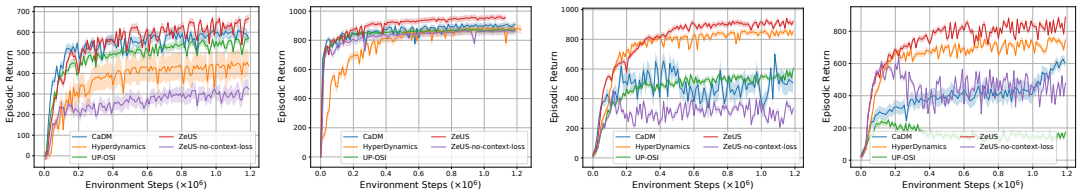

Figure 8: We compare the performance of the proposed ZeUS algorithm with *CaDM*, *UP-OSI*, *HyperDynamics* and *ZeUS-no-context-loss* algorithms on the evaluation environments (interpolation) for four families of tasks with different dynamics parameters.

C.3. Adapting and generalizing to environments with varying reward functions

In Figure 9 we show performance of ZeUS over a hyperparameter sweep for different values of aggregation operator and values of α_ψ for the Cheetah-run-v0 setup.

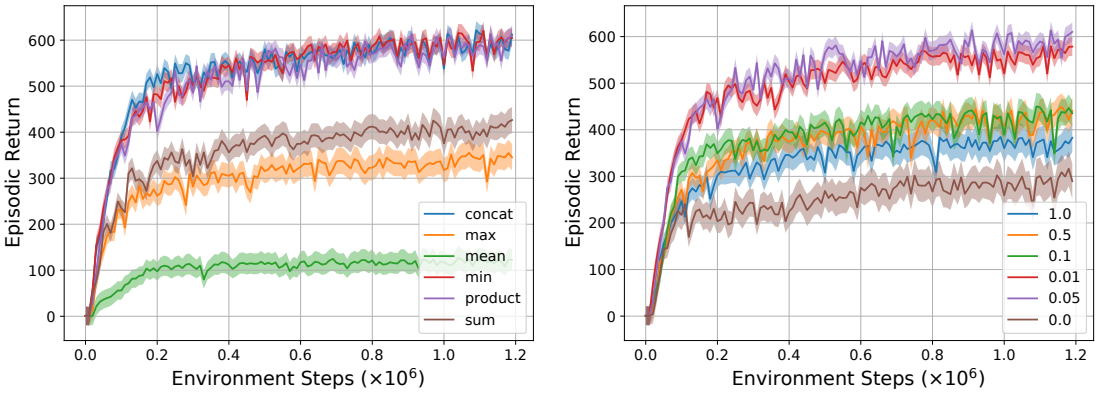

Figure 9: Performance of ZeUS on Cheetah-run-v0 task with varying values of the aggregation operator in the context enocder (in (a)) and varying values of α_ψ in (b).

Appendix D. Additional Theoretical Background

Bisimulation is a strict form of state abstraction, where two states are bisimilar if they are behaviorally equivalent. **Bisimulation metrics** (Ferns et al., 2011) define a distance between states as follows:

Definition 6 (Bisimulation Metric (Theorem 2.6 in Ferns et al. (2011))) *Let $(\mathcal{S}, \mathcal{A}, T, r)$ be a finite MDP and $\mathfrak{met}$ the space of bounded pseudometrics on $\mathcal{S}$ equipped with the metric induced by the uniform norm. Define $F : \mathfrak{met} \mapsto \mathfrak{met}$ by*

$$F(d)(s, s') = \max_{a \in \mathcal{A}} (|r(s, a) - r(s', a)| + \gamma W(d)(T^a_s, T^a_{s'})),$$

where $W(d)$ is the Wasserstein distance between transition probability distributions. Then F has a unique fixed point $\tilde{d}$ which is the bisimulation metric.

A nice property of this metric $\tilde{d}$ is that difference in optimal value between two states is bounded by their distance as defined by this metric. Zhang et al. (2021a) use bisimulation metrics to learn a representation space that is Lipschitz with respect to the MDP dynamics in the single task setting.

Why is the bisimulation metric useful? It turns out that the optimal value function has the nice property of being smooth with respect to this metric.

Lemma 1 (V^* is Lipschitz with respect to $\tilde{d}$ (Ferns et al., 2004)) *Let V^* be the optimal value function for a given discount factor γ. Then V^* is Lipschitz continuous with respect to $\tilde{d}$ with Lipschitz constant $\frac{1}{1-\gamma}$,*

$$|V^*(s) - V^*(s')| \leq \frac{1}{1-\gamma} \tilde{d}(s, s').$$

Proof in (Ferns et al., 2004). Therefore, we see that bisimulation metrics give us a Lipschitz value function with respect to $\tilde{d}$ with a Lipschitz constant $\frac{1}{1-\gamma}$.

Appendix E. Additional Theoretical Results and Proofs

Corollary 1 (1) *Let V^* be the optimal, universal value function for a given discount factor γ and context space $\mathcal{C}$. Then V^* is Lipschitz continuous with respect to d_{task} with Lipschitz constant $\frac{1}{1-\gamma}$ for any $s \in \mathcal{S}$,*

$$|V^*(s, c) - V^*(s, c')| \leq \frac{1}{1-\gamma} d_{task}(c, c').$$

Proof We construct a super-MDP $\mathcal{M}'$ by concatenating context and state spaces into a new state space $\mathcal{S}' : \mathcal{C} \times \mathcal{S}$. We can apply Lemma 1 to $\mathcal{M}'$ for any $s \in \mathcal{S}, c, c' \in \mathcal{C}$:

$$|V^*_{\mathcal{M}'}((s, c)) - V^*_{\mathcal{M}'}((s, c'))| \leq \frac{1}{1-\gamma} \tilde{d}_{\mathcal{M}'}((s, c), (s, c')). \tag{3}$$

By Definition 6 and Definition 4 we know that

$$\tilde{d}_{\mathcal{M}'}((s, c), (s, c')) \leq d_{\text{task}}(c, c').$$

So we can substitute the right hand side into Equation (3),

$$|V^*_{\mathcal{M}'}((s, c)) - V^*_{\mathcal{M}'}((s, c'))| \leq \frac{1}{1-\gamma} d_{\text{task}}(c, c').$$

Which is equivalent to the following statement:

$$|V^*(s, c) - V^*(s, c')| \leq \frac{1}{1-\gamma} d_{\text{task}}(c, c').$$

■

E.1. Value and Transfer Bounds

In this section, we provide value bounds and sample complexity analysis of the ZeUS approach. This analysis is similar to the one done in Zhang et al. (2021b), which focused on a multi-environment setting with different dynamics but same task. We first define three additional error terms associated with learning a $\epsilon_R, \epsilon_T, \epsilon_c$-bisimulation abstraction,

$$\begin{aligned}
\epsilon_R &:= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \big|R(o_1,a) - R(o_2,a)\big|,\\
\epsilon_T &:= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \big\|\Phi T(o_1,a) - \Phi T(o_2,a)\big\|_1,\\
\epsilon_c &:= \|\hat{c} - c\|_1.
\end{aligned}$$

ϵ_R and ϵ_T are intra-context constants and ϵ_c is an inter-context constant. ΦT denotes the *lifted* version of T, where we take the next-step transition distribution from observation space $\mathcal{O}$ and lift it to latent space $\mathcal{S}$. We can think of ϵ_R, ϵ_T as describing a new MDP which is close – but not necessarily the same, if $\epsilon_R, \epsilon_T > 0$ – to the original MDP. These two error terms can be computed empirically over all training environments and are therefore not task-specific. ϵ_c, on the other hand, is measured as a per-task error. Similar methods are used in Jiang et al. (2015) to bound the loss of a single abstraction, which we extend to the BC-MDP setting with a family of tasks.

Value Bounds. We first look at the single, fixed context setting, which can be thought of as the single-task version of the BC-MDP. We can compute approximate error bounds in this setting by denoting ϕ an (ϵ_R, ϵ_T)-approximate bisimulation abstraction, where

$$\begin{aligned}
\epsilon_R &:= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \big|R(o_1,a) - R(o_2,a)\big|,\\
\epsilon_T &:= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \big\|\Phi T(o_1,a) - \Phi T(o_2,a)\big\|_1,\\
\epsilon_c &:= \|\hat{c} - c\|_1.
\end{aligned}$$

ΦT denotes the *lifted* version of T, where we take the next-step transition distribution from observation space $\mathcal{O}$ and lift it to latent space $\mathcal{S}$.

Lemma 2 *Given an MDP $\bar{\mathcal{M}}$ built on a (ϵ_R, ϵ_T)-approximate bisimulation abstraction of Block MDP $\mathcal{M}$, we denote the evaluation of the optimal Q function of $\bar{\mathcal{M}}$ on $\mathcal{M}$ as $[Q^*_{\bar{\mathcal{M}}}]_{\mathcal{M}}$. The value difference with respect to the optimal $Q^*_{\mathcal{M}}$ is upper bounded by*

$$\big\|Q^*_{\mathcal{M}} - [Q^*_{\bar{\mathcal{M}}}]_{\mathcal{M}}\big\|_\infty \leq \epsilon_R + \gamma\epsilon_T \frac{R_{max}}{2(1-\gamma)}.$$

Proof From Lemma 3 in Jiang et al. (2015). ■

We now evaluate how the error in c prediction and the learned bisimulation representation affect the optimal $Q^*_{\bar{\mathcal{M}}_{\hat{c}}}$ of the learned MDP, by first bounding its distance from the optimal Q^* of the true MDP for a single-task.

Lemma 3 (Q error) *Given an MDP $\bar{\mathcal{M}}_{\hat{c}}$ built on a $(\epsilon_R, \epsilon_T, \epsilon_c)$-approximate bisimulation abstraction of an instance of a HiP-BMDP $\mathcal{M}_c$, we denote the evaluation of the optimal Q function of $\bar{\mathcal{M}}_{\hat{c}}$ on $\mathcal{M}$ as $[Q^*_{\bar{\mathcal{M}}_{\hat{c}}}]_{\mathcal{M}_c}$. The value difference with respect to the optimal $Q^*_{\mathcal{M}}$ is upper bounded by*

$$\left\|Q^*_{\mathcal{M}_c} - [Q^*_{\bar{\mathcal{M}}_{\hat{c}}}]_{\mathcal{M}_c}\right\|_\infty \leq \epsilon_R + \gamma(\epsilon_T + \epsilon_c)\frac{R_{max}}{2(1-\gamma)}.$$

Proof In the BC-MDP setting, we have a global encoder ϕ over all tasks, but the different transition distributions and reward functions condition on the context c. We now must incorporate difference in dynamics in ϵ_T and reward in ϵ_R. Assuming we have two environments with hidden parameters c_i, c_j, we can compute $\epsilon_T^{c_i,c_j}$ and $\epsilon_R^{c_i,c_j}$ across those two environments by joining them into a super-MDP.: For $\epsilon_T^{c_i,c_j}$:

$$\begin{aligned}
\epsilon_T^{c_i,c_j} &= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left\|\Phi T_{c_i}(o_1,a) - \Phi T_{c_j}(o_2,a)\right\|_1 \\
&\leq \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left(\left\|\Phi T_{c_i}(o_1,a) - \Phi T_{c_i}(o_2,a)\right\|_1 + \left\|\Phi T_{c_i}(o_2,a) - \Phi T_{c_j}(o_2,a)\right\|_1\right) \\
&\leq \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left\|\Phi T_{c_i}(o_1,a) - \Phi T_{c_i}(o_2,a)\right\|_1 + \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left\|\Phi T_{c_i}(o_2,a) - \Phi T_{c_j}(o_2,a)\right\|_1
\end{aligned}$$

For $\epsilon_R^{c_i,c_j}$ it is much the same:

$$\begin{aligned}
\epsilon_R^{c_i,c_j} &= \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left|R_{c_i}(o_1,a) - R_{c_j}(o_2,a)\right| \\
&\leq \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left(\left|R_{c_i}(o_1,a) - R_{c_i}(o_2,a)\right| + \left|R_{c_i}(o_2,a) - R_{c_j}(o_2,a)\right|\right) \\
&\leq \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left|R_{c_i}(o_1,a) - R_{c_i}(o_2,a)\right| + \sup_{\substack{a\in\mathcal{A},\\ o_1,o_2\in\mathcal{O},\phi(o_1)=\phi(o_2)}} \left|R_{c_i}(o_2,a) - R_{c_j}(o_2,a)\right|
\end{aligned}$$

Putting these together we get:

$$\epsilon_T^{c_i,c_j} + \epsilon_R^{c_i,c_j} \leq \epsilon_T + \epsilon_R + \|c_i - c_j\|_1$$

This result is intuitive in that with a shared encoder learning a per-task bisimulation relation, the distance between bisimilar states from another task depends on the change in transition distribution between those two tasks. We can now extend the single-task bisimulation bound (Lemma 2) to the BC-BMDP setting by denoting approximation error of c as $\|c - \hat{c}\|_1 < \epsilon_c$. ■

We can measure the generalization capability of a specific policy π learned on one task to another, now taking into account error from the learned representation.

Theorem 1 (2) *Given two MDPs $\mathcal{M}_{c_i}$ and $\mathcal{M}_{c_j}$, we can bound the difference in Q^π between the two MDPs for a given policy π learned under an $\epsilon_R, \epsilon_T, \epsilon_{c_i}$-approximate abstraction of $\mathcal{M}_{c_i}$ and applied to*

$$\left\|Q^*_{\mathcal{M}_{c_j}} - [Q^*_{\bar{\mathcal{M}}_{\hat{c}_i}}]_{\mathcal{M}_{c_j}}\right\|_\infty \leq \epsilon_R + \gamma\big(\epsilon_T + \epsilon_{c_i} + \|c_i - c_j\|_1\big)\frac{R_{max}}{2(1-\gamma)}.$$

This result clearly follows directly from Lemma 3. Given a policy learned for task i, Theorem 2 gives a bound on how far from optimal that policy is when applied to task j. Intuitively, the more similar in behavior tasks i and j are, as denoted by $\|c_i - c_j\|_1$, the better π performs on task j.