Rethinking Learning Dynamics in RL using Adversarial Networks

Ramnath Kumar 1,*, Tristan Deleu 1,2, Yoshua Bengio 1,2,3

Abstract

We present a learning mechanism for reinforcement learning of closely related skills parameterized via a skill embedding space. Our approach is grounded on the intuition that nothing makes you learn better than a coevolving adversary. The main contribution of our work is to formulate an adversarial training regime for reinforcement learning with the help of entropy-regularized policy gradient formulation. We also adapt existing measures of causal attribution to draw insights from the skills learned. This would facilitate easier re-purposing of skills for adaptation to different environments and tasks. Our experiments demonstrate that the adversarial process leads to a better exploration of multiple solutions and understanding the minimum number of different skills necessary to solve a given set of tasks.

1 Introduction

Recent years have seen tremendous progress in methods for reinforcement learning with the rise of “Deep Reinforcement Learning” (DRL; 31). In the field of robotics, DRL holds the promise of automatically learning flexible behaviors end-to-end while dealing with multidimensional data as mentioned in [2]. The level has risen so much that our algorithms are capable of learning policies that can defeat human professionals in many different games such as Chess, Go, and many others (e.g. by AlphaZero [43]), and even more complicated games such as Dota-2 (OpenAI, as presented in [4]), amongst many other tasks such as robot control [19, 29, 32], to meta-learning [48].

Despite this recent progress, the predominant paradigm remains to train straightforwardly, only to learn what works and not understand what fails. This paper draws intuition from the statement that nothing makes you learn better than coevolving adversaries. However, the fact that an adversarial training regime has not been proposed earlier is understandable for many reasons; Why would we want to teach the model to fail? What advantages would the algorithm yield? Would there be any guarantees for convergence in such a twisted model? Some approaches do try to create adversarial examples to make the models better suited for outliers [37, 7]. Our work is novel in that we wish for the model to learn what does not work, leading to better exploration and more accessible domain adaptation to unseen tasks not by changing the tasks or rewards but by letting the same embedding network learn multiple skill distributions.

We aim to take a step in this direction, and propose a novel adversarial training regime for learning tasks in the reinforcement learning domain. Our method learns “manipulation skills” that are continuously parameterized in an embedding space. We also show advantages in performance in a given environment and adaptation of skills to unseen environments. To learn skills, we take advantage of latent variables – an essential tool in the probabilistic modeling literature for discovering structure in data. We can draw causal insights on these latent variables, and successfully determine the effect of each latent attribute on the goal or task at hand.

* Work done during an internship at Mila; Correspondence author ramnathkumar181@gmail.com. 1Mila, Québec Artificial Intelligence Institute. 2Université de Montréal. 3CIFAR, IVADO.

Preprint. Under review.
The main contribution of our work is an adversarial training regime for on-policy optimization using concepts from game theory literature, enabling us to learn distinct latent for each task and an efficient framework to approximate the average causal effect of each embedding on the reward obtained on the task in the on-policy domain. We further extend our framework, adapt it to off-policy learning which is crucial for potential applications of this method to real-world environments. We show how stochastic latent variables trained using an adversarial training regime can be meaningfully incorporated into policies by treating them the same way as auxiliary variables in parametric variational inference [21, 30, 41, 38]. The resulting policy can model complex correlation structure and multi-modality in action space – capable of knowing what actions are right and what is wrong. This provides an additional boost in the explorability of the agent and leads to the discovery of more efficient skills and trajectories. Although the model seems to be counter-intuitive and difficult to train, we demonstrate the effectiveness of our method on several simulated environments such as Point mass, 2D navigation, and robotic manipulation tasks from Meta-World. With our proposed causal attribution framework, we can also show that the model can learn multiple solutions with the minimum number of distinct skills necessary for the given set of tasks. We also show that the model trained using the adversarial training regime leads to better generalization and more accessible domain adaptation to unseen and complex tasks, even in sparse rewards. Some of the limitations of our approach is briefly described in Appendix E.

2 Related Works

This section describes prior studies that are broadly related to the adversarial training regime of skills. Below, we divide prior research under two prominent subheadings: (i) Adversarial training regime and (ii) Multi-Task reinforcement learning.

2.1 Adversarial Training Regime

Adversarial training regime methods have been predominantly used in minimax objective problems, including [18], which leverages adversarial examples to train more robust classifiers and [17, 11] which uses an adversarial loss function for a discriminator to train a generative model.

In reinforcement learning, learned policies should be robust to uncertainty and parameter variation to ensure predictable behavior. Furthermore, learning policies should employ safe and effective exploration with improved sample efficiency to reduce the risk of costly failure. These issues have long been recognized and studied in reinforcement learning [47, 16]. However, these issues are exacerbated in deep RL using neural networks, which, while more expressible and flexible, often require more data to train and produce potentially unstable policies. In [36], two supervised agents were trained, with one acting as an adversary for self-supervised learning, which showed an improvement in robot grasping. Other adversarial multiplayer approaches have been proposed, including [23]. However, these approaches require the training of two agents. Inspired by these, [23] proposed an approach to learn two policies for a single agent, one as protagonist and the other as an adversary. However, this requires hard-coding parts of the agent where an adversarial force is applied to fool the model. Although these approaches lead to a more robust model, our goal in this paper is slightly different.

Our goal is to learn a model that is not just robust but also capable of understanding correct and wrong actions through increased explorability. Instead of performing an adversarial training regime at a policy level and learning two policies, we perform the regime at the skill level, allowing better exploration. Since the same policy network is used in both cases, the policy understands the right actions and the effect of wrong actions, giving a sense of increased awareness and understanding of the environment.

2.2 Multi-Task reinforcement learning

The idea of learning multiple tasks at once in reinforcement learning is quite common with the intuition that more tasks would suggest higher diversity and hence better generalization in downstream tasks. This property is best showcased in the work [13], where they learn diverse skills without any reward function. Furthermore, sequential learning and the need to retain previously known skills has always been a focus [40, 26]. In the space of multi-task reinforcement learning with neural networks,
[44] proposed a framework that allows sharing of knowledge across tasks via a task agnostic prior. Similarly, [5] make use of off-policy learning to learn about a large number of different tasks while following a primary task. Other approaches [10, 9] propose architectures that can be reconfigured easily to solve a variety of tasks. Subsequently, [14] use meta-learning to acquire skills that can be fine-tuned effectively.

The use of latent variables and entropy constraints to induce diverse skills have been considered before [8, 12] albeit in a different framework and without using neural network function approximators. Further, [27] used the options framework to learn transferable options using the so-called agent space. Inspired by these ideas, [21] introduces a skill embedding learning method that uses deep reinforcement learning techniques and can concisely represent and reuse skills. Their works draws on a connection between entropy-regularized reinforcement learning and variational inference literature [45, 46, 34, 28, 39, 15]. The notion of latent variables in policies has been previously explored by works such as controllers [22], and options [3]. The auxiliary variable perspective introduces an informative-theoretic regularizer that helps the inference model produce more versatile behaviors. Learning these versatile skills has been previously explored by [20], which learns an energy-based, maximum entropy policy via Q-learning algorithm, and [42] which uses an entropy regularized reinforcement learning policy. [21] uses a similar entropy-regularized reinforcement learning and latent variables but differs in the algorithmic framework. In this work, we use a similar algorithmic framework; although their approach uses a more standard reinforcement learning paradigm, we learn our policy using an adversarial training regime. In addition, we also extend our method to a more sample-efficient off-policy setup, which is crucial for potential applications of this method to real-world environments.

3 Preliminaries

Before we delve into the details of our adversarial training regime-based algorithm ATE-PPO, we first outline our terminology, standard reinforcement learning setting, and two-player adversarial games from which our paper is inspired.

3.1 Standard reinforcement learning in MDPs

We study reinforcement learning in Markov Decision Processes (MDPs). We denote \( s \in \mathbb{R}^S \) as the continuous state of the agent; \( a \in \mathbb{R}^A \) as the action vector and \( p(s_{t+1} | s_t, a_t) \) as the probability of transitioning from state \( s_t \) to \( s_{t+1} \) when executing action \( a_t \). Actions are drawn from a policy distribution \( \pi_\theta(a | s) \), with parameters \( \theta \). This work uses a Gaussian distribution whose mean and diagonal covariance are parameterized via a neural network. At every step, the agent receives a scalar reward \( r(s_t, a_t) \) and we consider the problem of maximizing the sum of discounted rewards:

\[
E_{\tau} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right].
\]

We use TE-PPO [21] as our baseline since it also makes use of latent skills for learning policy and serves as the perfect baseline to showcase the effect of adversarial training regime in reinforcement learning, keeping all other things constant.

3.2 Two-player adversarial games

The adversarial setting we propose can be expressed as a two-player \( \gamma \)-discounted non-zero-sum Markov Game. Instead of learning the policy directly in an adversarial manner, we choose to learn the skills used by the policy in an adversarial manner. This game MDP is similar to the previous, with an added variable from the adversary – a new skill that reduces the average discounted return on a given task. Player 1 (Encoder + policy) will try to act as a protagonist and solve the given task at hand, whereas Player 2 (Encoder alone) will serve as an adversary and learn skills that best fool the protagonist. Note that while Player 2 acts, the policy network is frozen, and only the embedding network is learned. We call our algorithm Adversarial TE-PPO (or ATE-PPO) and describe our model in more detail in Section 4. A brief overview of our algorithm is presented in Algorithm 1. Our end-objective function works in the minimax domain as discussed in the subsequent section and differentiates our work from TE-PPO. Their work operates in the straightforward RL domain and ours in the adversarial domain (involving fooling the agent and learning efficient skills).
Input: Data sampling mechanism $T$, Data $D$, number of episodes $N$, Encoder Networks $E$, Policy Network $\pi$, Number of Adversarial steps $A$, and Number of Protagonist steps $P$.

$n \leftarrow N$;
$t_i \leftarrow \text{OneHot}(i)$; // Create one hot vectors of size $k$
$t \leftarrow \{t_1, t_2, ..., t_k\}$; // Create Task Embedding

while $n < N$ do

$n \leftarrow n + 1$;

$\{T_1, T_2, ..., T_k\} \leftarrow T(D)$; // Sample $k$ tasks from data distribution.
$a \leftarrow 0$;
$p \leftarrow 0$;
$\pi^* \leftarrow \pi$; // Fix the Policy Network.

while $a < A$ do

$a \leftarrow a + 1$;

$\min_{E} O_{V}(E(t), \pi^*(E(t)))$; // Learn adversarial skills

end

$E^* \leftarrow E$; // Fix the Encoder Network.

while $p < P$ do

$a \leftarrow a + 1$;

$max_{\pi} O_{V}(E^*(t), \pi(E(t)))$; // Learn protagonist skills

end

end

Algorithm 1: Adversarial Training Regime

4 Adversarial Training Regime in RL

Our approach is comprised of three different networks which are trained in an adversarial training regime: an Encoder network ($E$), a Policy network ($\pi$), and an Inference network ($I$). The Encoder tries to learn a skill embedding - something unique for each task at hand, and the Policy network uses this embedding to select actions maximizing its reward. The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons.

To learn the skill embedding $z$, we represent a mapping between the one-hot encoded task embedding $t$ to skill space $z$ as $z = E(t; \theta_e)$; where $E$ is a differentiable function represented by a multilayer perceptron with parameters $\theta_e$ and $z$ is the skill required for achieving task $t$. To take advantage of these skill embeddings, we define a second multilayer perceptron $\pi(z; \theta_\alpha)$ that helps the agent learn the actions to take and maximize its reward. Note that similar to [21], we also allow the encoder to learn in this step to facilitate learning optimal skills for the given task. To achieve this training regime, our model plays a two-player minimax game with value function $V(E, \pi)$ and value objective $O_{V}(E, \pi)$:

$$V(E, \pi) = \min_{E} \max_{\pi} O_{V}(E, \pi),$$

where

$$O_{V}(E, \pi) = E_{\pi, p_0, t \in T} [Q_\pi(s, a; t)] - \alpha E_{\pi, t \in T} [\mathcal{H}(z) + \mathcal{H}(t) - \mathcal{H}(t|z)]$$

where $p_0(s_0)$ is the initial state distribution, $\alpha$ is a weighing term – a trade-off between deceiving the agent network, and minimizing the entropy terms denoted by $\mathcal{H}(.)$. Furthermore, the Q-function $Q_\pi(s, a, t)$ is defined as:

$$Q_\pi(s, a, t) = \sum_{i=0}^{\infty} \gamma^i r_i(s_i, a_i) + \alpha_0 \mathcal{H}[\pi(a_i | s_i, t)]$$

where action $a_i \sim \pi(\cdot | s, t)$, and the new state is $s_{i+1} \sim p(s_{i+1} | a_i, s_i)$, and $\alpha_0$ is a constant. The first half of the equation defined by $\sum_{i=0}^{\infty} \gamma^i r_i(s_i, a_i)$ is the discounted expected returns and is common to our generic reinforcement learning algorithms. The latter is entropy regularization term defined by $\sum_{i=0}^{\infty} \gamma^i \mathcal{H}[\pi(a_i | s_i, t)]$ which is conventionally applied to many policy gradient schemes, with the critical difference that it takes into account not only the entropy term of the current, but also future actions. This approach has been used before by [21], but with a different objective function, and trained in a straightforward reinforcement learning fashion, without any adversarial training dynamics. Note that although we are trying to solve a minimax equation with $\pi$ and $E$, in practice,
we consider the embedding network linked to the policy, and both should be learnt together during the learning phase of the protagonist.

To apply this entropy regularization to our setting of latent variables (skill embeddings), some extra mathematical rigor is required. Borrowing from the toolkit of variational inference [1], we can construct a lower bound on the entropy term from Equation 2 as (see Appendix B) such that:

$$E_{\pi,p_0,t \in T} [Q^\pi_t(s, a, t)] = E_{\pi(z|s,t)} [Q^\pi_t(s, a; z, t)] + \alpha_1 E_{t \in T} H[p(z|t)]$$

where,

$$Q^\pi_t(s, a; z, t) = \sum_{i=0}^{\infty} \gamma^i \hat{r}(s_i, a_i, z, t)$$

Here, $s_{i+1} \sim p(s_{i+1}|a_i, s_i)$, and $\hat{r}(s_i, a_i, z, t) = \left[r_t(s_i, a_i) + \alpha_2 \log[\mathbb{I}(z|a_i, s_i^H)] + \alpha_3 H[\pi(a|s, z)]\right]$. Note that this is where the inference network $I$ comes into play. Furthermore, $H$ is the history of states saved in the buffer used for the inference. From the above equation, we simplify our objective function bound for skill learning as shown in Appendix D. This simplification will also remove the redundant overhead, we simplify the objective presented in Equation 3 as shown in Appendix D. This

$$O_V(\mathcal{E}, \pi) = [Q^\pi_t(s, a; z, t)] - \alpha E_{t \in T} [H(z) - H(z|t) + H(t) - H(t|z)]$$

$$= [Q^\pi_t(s, a; z, t)] - \alpha E_{t \in T} [I(z|t) + I(t|z)]$$

$$= [Q^\pi_t(s, a; z, t)] - \alpha E_{t \in T} [\text{JSD}(z, t)]$$

Intuitively, the resulting objective function bound meets the following desiderata:

- **Discounted Returns** $r_t(s_i, a_i)$: The discounted returns objective is used to learn an policy successful in producing high rewards for a given task and environment.
- **Cross Entropy** $\log[\mathbb{I}(z|a_i, s_i^H)]$: This term encourages different embedding vectors $z$ to have different effects in terms of executed actions and visited states. Intuitively, it will be high when we can predict $z$ from the resulting $a, s^H$, where $H$ holds the same meaning as defined earlier. Note that $\mathbb{I}$ here denotes the inference network.
- **Entropy of the policy conditioned on the embedding** $H[\pi(a|s, z)]$: This term aims to cover the embedding space with different skills.
- **Entropy of the embedding given task** $H(z|t)$: Minimizing the entropy of the embedding given a task ensures that the skill embedding has a sharp peak and is deterministic.
- **Entropy of the embedding** $H(z)$: We maximize the entropy of the skill embedding to ensure that our algorithm covers all the skills in the latent space.
- **Entropy of task** $H(t)$: Since the tasks are sampled uniformly, we maximize the entropy and are already at the optimal solution.
- **Entropy of task given embedding** $H(t|z)$: Similar to $H(z|t)$, we want the prediction of $t$ given $z$ to be deterministic and to have a narrow peak. However, to avoid any computational overhead, we simplify the objective presented in Equation 3 as shown in Appendix D. This simplification will also remove the redundant $H(t)$ computation.

In this paper, we aim to answer the following questions:

**i) Can our method learn versatile skills even in the adversarial training regime?** We aim to learn a skill embedding space, in which different embedding vectors that are “close” to each other in the embedding space correspond to distinct solutions to the task. We enforce this property by minimizing the entropy of skill space given a task. We also support this using our experiments on point mass and 2-D navigation tasks.

**ii) Does the training regime aid better exploration and learning in agents?** We aim for the agent to better explore the environment to learn and leverage information from its surroundings. Our adversarial training regime, which tries to fool the model, should lead to more exploration as more steps are taken to understand exactly what actions fail to accomplish a task. We also support this using our experiments on the Point Mass and constrained 2-D navigation environment. Furthermore, we also show that our ATE-PPO algorithm outperforms the standard TE-PPO algorithm across all experiments in this paper.
We highlight that the above derivation and steps also hold when the task id is constant, i.e., for the

$$
\hat{\text{wherein,}}
$$

$$
N
$$

we bootstrap the infinite sum after

$$
Q
$$

which quickly allows us to propagate entropy augmented rewards across multiple time-steps while

$$
Q\text{-function}
$$

that depends on both states

$$
\text{we formulate the off-policy perspective of our algorithm. We start with the notion of a}
$$

lower-bound

$$
\text{probability}
$$

of a replay buffer

$$
B
$$

(containing full trajectory execution traces including states, actions, task id, and reward) that is inherently filled during training. In conjunction with these trajectory traces, we also store the probabilities of each selected action and denote them with the behavior policy probability

$$
b(a|z, s, t)
$$

and the behavior probabilities of the embedding

$$
b(z|t)
$$

given the necessary conditions are guaranteed. In practice, we cannot sequentially train the two

network training regime, essentially showing that the training criterion converges appropriately

following sections in Appendix C present a more detailed theoretical analysis of our adversarial

training regime, instead of introducing a \( \lambda \) parameter as in the original paper. Equipped with this Q-function, we can update the policy and embedding network parameters without requiring additional environment interactions by optimizing the same objective as Eq. 3.

We highlight that the above derivation and steps also hold when the task id is constant, i.e., for the

regular reinforcement learning setting, rather than the meta-reinforcement learning setting. The

following sections in Appendix C present a more detailed theoretical analysis of our adversarial

network training regime, essentially showing that the training criterion converges appropriately
given the necessary conditions are guaranteed. In practice, we cannot sequentially train the two

networks. We must implement the game using an iterative back and forward approach. Optimizing \( \mathcal{E} \) to completion would lead to overfitting, and the Encoder would not learn the adequate skill embedding
to be used by the policy network. Instead, we alternate between \( k \) steps of optimizing the protagonist
and one stage of optimizing the adversary. This results in the protagonist being maintained near
its optimal solution, so long as adversarial skills change slowly enough. It might seem counter-
intuitive to minimize the discounted rewards by the embedding function. However, as long as the

\( v \) and one stage of optimizing the adversary. This results in the protagonist being maintained near
its optimal solution, so long as adversarial skills change slowly enough. It might seem counter-
intuitive to minimize the discounted rewards by the embedding function. However, as long as the

iii) Is our model interpretable? Given the state and action trace of an executed skill, we would
also like to identify the embedding vector (skill) that gave rise to the solution. This property would
allow us to re-purpose the embedding space for solving new tasks, especially in the case of meta-
learning. We propose a simple approximation to compute the Average Causal Effect (ACE) of the
rewards given the latent skill to address this requirement. Our proposed framework is discussed
further in Appendix A.

iv) Does the training regime lead to better generalization? We desire an embedding space in
which solutions to different and potentially orthogonal tasks can be represented. We substantiate this
property with our proof of mutual exclusivity between tasks \( t \) and skill space \( z \). We also support this
using our experiments on transfer learning on the Meta-World dataset as discussed in Section 5.3.

Extending to Off-Policy setting While the objective presented in Equation 3 could be optimized
directly in an on-policy setting, we would also like to extend this training regime to obtain a data-
efficient, off-policy algorithm that could be applied to a natural robotic system in the future. In this
section, estimate the discounted sums from Equation 3 from previously gathered data by learning
a Q-value function, yielding an off-policy algorithm. Similar to [21], we assume the availability
of a replay buffer \( B \) (containing full trajectory execution traces including states, actions, task id, and reward) that is inherently filled during training. In conjunction with these trajectory traces, we also store the probabilities of each selected action and denote them with the behavior policy probability

$$
b(a|z, s, t)
$$

and the behavior probabilities of the embedding

$$
b(z|t)
$$

given the necessary conditions are guaranteed. In practice, we cannot sequentially train the two

network training regime, essentially showing that the training criterion converges appropriately

following sections in Appendix C present a more detailed theoretical analysis of our adversarial

training regime, instead of introducing a \( \lambda \) parameter as in the original paper. Equipped with this Q-function, we can update the policy and embedding network parameters without requiring additional environment interactions by optimizing the same objective as Eq. 3.

We highlight that the above derivation and steps also hold when the task id is constant, i.e., for the

regular reinforcement learning setting, rather than the meta-reinforcement learning setting. The

following sections in Appendix C present a more detailed theoretical analysis of our adversarial

network training regime, essentially showing that the training criterion converges appropriately
given the necessary conditions are guaranteed. In practice, we cannot sequentially train the two

networks. We must implement the game using an iterative back and forward approach. Optimizing \( \mathcal{E} \) to completion would lead to overfitting, and the Encoder would not learn the adequate skill embedding
to be used by the policy network. Instead, we alternate between \( k \) steps of optimizing the protagonist
and one stage of optimizing the adversary. This results in the protagonist being maintained near
its optimal solution, so long as adversarial skills change slowly enough. It might seem counter-
intuitive to minimize the discounted rewards by the embedding function. However, as long as the

\( v \) and one stage of optimizing the adversary. This results in the protagonist being maintained near
its optimal solution, so long as adversarial skills change slowly enough. It might seem counter-
intuitive to minimize the discounted rewards by the embedding function. However, as long as the
hyperparameter $\alpha$ is set such that the Jensen-Shannon Divergence objective prioritizes the discounted rewards criterion. With this assumption, we can ensure that minimizing the expected returns would merely act as an implicit noise [24].

5 Experimental Results

We evaluate our approach in two domains in simulation: simple environments such as point mass task, 2-D navigation task, and a set of challenging robot manipulation tasks from Meta-World. We consider the Gaussian embedding space for all experiments for both algorithms (ATE-PPO and TE-PPO).

5.1 Learning versatile skills

We highlight this property of learning versatile skills from our experiments on relatively more straightforward environments such as the PointMass environment and 2-D Navigation environment.

**PointMass Environment** Similar to [20], we present a didactic example of multi-goal point mass tasks that demonstrate the variability of solutions that our method can discover. In this experiment, we consider a case where four goals are located around the initial location, and each of them is equally important to the agent. This leads to a situation where multiple optimal policies exist for a single task. In addition, this task is challenging due to the sparsity of the rewards – as soon as one solution is discovered, it becomes exceedingly difficult to keep exploring other goals. Due to these challenges, most existing DRL approaches would be content with finding a single solution. Furthermore, even if a standard policy gradient approach discovered multiple goals, it would have no incentive to represent various solutions. However, with the added goal to maximize $\mathcal{H}(z)$, our ATE-PPO algorithm should lead to better exploration and learning different skills.

![Learning Curve](a) Learning Curve

![Trajectories](b) TE-PPO
![Trajectories](c) ATE-PPO

Figure 1: Figure 1(a) depicts the Average return on PointMass Environment using TE-PPO and ATE-PPO algorithm. The above illustration shows that the adversarial training regime does offer a boost in performance in the PointMass environment. Figure 1(b) and Figure 1(c) show the resulting trajectories by the agent trained on TE-PPO and ATE-PPO respectively.

To better understand the skill learned by our model, we use the causal framework proposed in Appendix A to compute the average causal attribution of each feature of the skill vector. The average importance of each component is shown in Figure 2. This result is interesting because the feature $z_2$ consistently does not contribute in any way towards the behavior of the agent. Although the latent space is ample, and the model has sufficient capacity, our policy network can learn only the minimum number of skills required to solve the task. Furthermore, we discuss the ACE plots in more detail in Appendix G.1.

**2-D Navigation Environment** Furthermore, we also experiment on the 2-D navigation task [14] with a slight modification. At each time step, the 2D agent takes action (its velocity, clipped in $[-0.1, 0.1]$) and receives a penalty equal to its L2 distance to the goal position (i.e., the reward is “−distance”). However, we restrict the agent’s movement in the beginning to a small corridor, such that the state space is clipped between $[-0.2, 0.2]$. This paper samples five tasks from this environment for our experiments. We find that the ATE-PPO model can successfully learn the task regardless and achieve better performance than the TE-PPO algorithm as depicted in Table 1. We notice that our ATE-PPO model successfully learns skills that are diverse in the future and not the...
5.2 Learning optimal representations

Next, we evaluate whether we can learn better representations with the help of the adversarial training regime. We create a pool of tasks from Meta-World with an overlapping skill set for most tasks for this problem. In our experiment, we set this overlapping skill to be the act of “pushing”. With this constraint, we create an environment MT5 by selecting five tasks for training. We choose the following five tasks for our MT5 environment: Push, Open window, Close window, Open drawer, and Close drawer. The multi-task evaluation here tests the ability to learn multiple tasks simultaneously without accounting for generalization to new tasks. This task helps evaluate the efficiency of the skills learned by the model. We show that ATE-PPO significantly improves TE-PPO’s performance without hyperparameter tuning and using common parameters across both experiments. TE-PPO achieves an average success rate of $0.2 \pm 0.01$, while our ATE-PPO achieves an average success rate of $0.41 \pm 0.08$. Furthermore, we believe the model’s performance could be significantly improved with hyperparameter-tuning since the model reaches a performance as high as 0.6 on some runs.

We summarize more detailed results of the causal analysis of latent in Appendix G.3.

5.3 Generalizing to unseen environments

Next, we evaluate whether our model can adapt to a new unseen task with minimum adaptation steps. We use our pre-trained model from the previous section for this task, trained on the MT5 environment. We selected a diverse set of tasks for this experiment with varying degrees of domain shift:

- **Push Wall**: The goal of this task is similar to the initial Push task from MT5, but we now also have to bypass a wall to reach the goal.
- **Coffee Button**: The goal of this task is to press a button on the coffee machine. We randomize the position of the coffee machine to avoid overfitting.
Figure 4: Figure 4(a) Average success rate on MT5 Environment using TE-PPO and ATE-PPO algorithm. The above illustration shows that the adversarial training regime does offer a boost in performance in the MT5 environment.

Table 1: Average return on simulated environments over 5 seeds.

| ENVIRONMENT     | TE-PPO        | ATE-PPO       |
|-----------------|---------------|---------------|
| 2D NAVIGATION   | -170.03 ± 47.98 | -102.39 ± 44.44 |
| PUSH WALL       | 2.54 ± 0.02   | 2.69 ± 0.18   |
| COFFEE BUTTON   | 37.21 ± 0.82  | 80.91 ± 2.80  |
| PUSH BACK       | 0.85 ± 0.01   | 0.89 ± 0.05   |
| FAUCET OPEN     | 474.41 ± 269.73 | 584.29 ± 237.43 |

- **Push Back**: The goal of this task is to push the same object backward, similar to the Push task from MT5. We also randomize puck positions to avoid overfitting.

- **Faucet Open**: The goal of the task is to turn on the faucet. Specifically, we would like to rotate the faucet counter-clockwise and randomize faucet positions to avoid overfitting.

We run the agent for all the above domain adaptation tasks for ten epochs. Although the domain adapted code is not sufficiently trained to solve the task perfectly, we find that our pre-trained ATE-PPO model outperforms the TE-PPO across majority of target tasks (if not, is competitive with), as shown in Table 1. Furthermore, for better comparison, we run both models in the standard reinforcement learning setting without any adversarial training regime.

6 Conclusion

We present an adversarial training regime that learns manipulation skills that are continuously parameterized in a skill embedding space. Our algorithm takes advantage of these skills and creates an adversary at the skill level instead of the policy level. The skills are learned by taking advantage of the increased explorability of known variables and exploiting a connection between reinforcement learning, variational inference, adversarial networks, and minimax algorithm from game theory. We derived an entropy regularized policy gradient formulation for reinforcement learning and demonstrated its robustness on multiple environments. Primarily, we showed that our model could learn skills that diverge in the future rather than the present. We also introduce a methodology for computing ACE. Our off-the-shelf implementation is model-agnostic and could help understand the causal structure of the skills. This would help in re-purposing skills and using them efficiently in ood environments. Our experiments with the help of ACE also indicate that even in the presence of enough capacity, our model is capable of discovering multiple solutions and is capable of learning the minimum number of distinct skills, as well as latents that are necessary to solve a given task. Furthermore, the learned skills are more adaptable and make for better cross-domain adaptation than the traditional reinforcement learning approach. We believe that the proposed framework is more ubiquitous than just reinforcement learning, and could be adapted to other domains such as supervised learning, representation learning, and many others.
Reproducability Statement

In this paper, we work with three different datasets, which are all open-sourced. Furthermore, we experiment with two different models - TE-PPO and ATE-PPO. The TE-PPO model was run after reproducing from their open-source code\(^1\). Additional details about setting up these models, and the hyperparameters are available in Appendix H. Our source code is made available for additional reference \(^2\).

Acknowledgements

We would like to thank Sony Corporation for funding this research through the Sony Research Award Program.

References

[1] David Barber Felix Agakov. The im algorithm: a variational approach to information maximization. Advances in neural information processing systems, 16(320):201, 2004.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866, 2017.

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 31, 2017.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680, 2019.

[5] Serkan Cabi, Sergio Gómez Colmenarejo, Matthew W Hoffman, Misha Denil, Ziyu Wang, and Nando Freitas. The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. In Conference on Robot Learning, pages 207–216. PMLR, 2017.

[6] Aditya Chattopadhyay, Piyushi Manupriya, Anirban Sarkar, and Vineeth N Balasubramanian. Neural network attributions: A causal perspective. In International Conference on Machine Learning, pages 981–990. PMLR, 2019.

[7] Tong Chen, Jiqiang Liu, Yingxiao Xiang, Wenjia Niu, Endong Tong, and Zhen Han. Adversarial attack and defense in reinforcement learning-from ai security view. Cybersecurity, 2(1):1–22, 2019.

[8] Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In Artificial Intelligence and Statistics, pages 273–281. PMLR, 2012.

[9] Misha Denil, Sergio Gómez Colmenarejo, Serkan Cabi, David Saxton, and Nando de Freitas. Programmable agents. arXiv preprint arXiv:1706.06383, 2017.

[10] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In 2017 IEEE international conference on robotics and automation (ICRA), pages 2169–2176. IEEE, 2017.

[11] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. arXiv preprint arXiv:1606.00704, 2016.

[12] Felix End, Riad Akour, Jan Peters, and Gerhard Neumann. Layered direct policy search for learning hierarchical skills. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 6442–6448. IEEE, 2017.

[13] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. arXiv preprint arXiv:1802.06070, 2018.

\(^1\)https://github.com/rlworkgroup/garage
\(^2\)https://github.com/RamnathKumari81/Adversarial-Learning-Dynamics-in-RL
[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning, pages 1126–1135. PMLR, 2017.

[15] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. arXiv preprint arXiv:1512.08562, 2015.

[16] Javier Garcia and Fernando Fernández. A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research, 16(1):1437–1480, 2015.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.

[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572, 2014.

[19] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In International conference on machine learning, pages 2829–2838. PMLR, 2016.

[20] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In International Conference on Machine Learning, pages 1352–1361. PMLR, 2017.

[21] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In International Conference on Learning Representations, 2018.

[22] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. arXiv preprint arXiv:1610.05182, 2016.

[23] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. arXiv preprint arXiv:1603.01121, 2016.

[24] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. arXiv preprint arXiv:1808.06670, 2018.

[25] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un)reliability of saliency methods. In Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, pages 267–280. Springer, 2019.

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences, 114(13):3521–3526, 2017.

[27] George Dimitri Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In IJCAI, volume 7, pages 895–900, 2007.

[28] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. Advances in neural information processing systems, 26:207–215, 2013.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[30] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. In International conference on machine learning, pages 1445–1453. PMLR, 2016.
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. *Advances in Neural Information Processing Systems*, 28:3132–3140, 2015.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G Bellemare. Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*, 2016.

Gerhard Neumann et al. Variational inference for policy search in changing situations. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 817–824, 2011.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Lerrel Pinto, James Davidson, and Abhinav Gupta. Supervision via competition: Robot adversaries for learning tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1601–1608. IEEE, 2017.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826. PMLR, 2017.

Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International Conference on Machine Learning*, pages 324–333. PMLR, 2016.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Twenty-third international joint conference on artificial intelligence*, 2013.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226. PMLR, 2015.

John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

Emanuel Todorov. General duality between optimal control and estimation. In *2008 47th IEEE Conference on Decision and Control*, pages 4286–4292. IEEE, 2008.

Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

Kemin Zhou and John Comstock Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.
A Causal Perspective into RL

The use of latent embeddings as skills can aid study the causality of the skill upon the task at hand. In this work, we propose a simple framework to compute the Average Causal Attribution of the skills on the task. The Average Causal Effect (ACE) of a variable $x$ on another random variable $y$ is commonly defined as:

$$ACE_{do(x_i = \alpha)}^{y} = \mathbb{E}[y|do(x_i = \alpha)] - baseline_x,$$

The ACE requires the computation of two quantities: the interventional expectation ($\mathbb{E}[y|do(x_i = \alpha)]$) and the baseline ($baseline_x$). Here, $do$ corresponds to the do-operation [35] of fixing a variable $x_i$ to a specific value $\alpha$.

**Interventional Expectation** The interventional expectation is a function of $x_i$ as all other variables marginalized out. By definition:

$$\mathbb{E}[y|do(x_i = \alpha)] = \int_{y} yp(y|do(x_i = \alpha))dy$$

Naively, evaluating the interventional expectation would involve sampling all other input features from the empirical distribution, keeping feature $X_i = \alpha$, and then averaging the output values. However, this computation assumes that the input features do not cause each other and would be time-consuming to run for the entire training data. Furthermore, we also assume that the features are d-separated: given an intervention on a particular variable, the probability distribution of all other neurons does not change, i.e. $p(x_j|do(x_i = \alpha)) = p(x_j)$, if $i \neq j$. Since the latent learned by our models are time-agnostic, our assumption is justifiable. To make the computation feasible, we make use of first-order Taylor expansion of the causal mechanism $f(x|do(x_i = \alpha))$ around the mean vector $\mu = [\mu_1, \mu_2, \cdots, \mu_k]^T$. is given by:

$$f(x|do(x_i = \alpha)) \approx f(\mu|do(x_i = \alpha)) + \nabla f(\mu|do(x_i = \alpha))^\top (x - \mu|do(x_i = \alpha))$$

Taking expectation on both sides (marginalizing over all other input neurons), we arrive at the Intervventional expectation:

$$\mathbb{E}[f(x|do(x_i = \alpha))] \approx f(\mu|do(x_i = \alpha)),$$

where $f(.)$ is the average reward for a given task, $x$ is the input vector or the latent embedding, and $\mu$ is the mean of input vectors when $x_i = \alpha$. Note that the first-order term vanishes since $\mathbb{E}(x|x_i = \alpha) = \mu$. This is extremely important since computing the gradient through a step in a reinforcement learning environment is impossible, since the transition function is in general non-differentiable. To make it differentiable, additional approximations and workarounds would be needed to achieve the same. Hence, we do not extend the same Taylor approximation to the second-order or higher for the same reason. Such approximations of deep non-linear neural networks via Taylor’s expansion have been explored before in the context of causality [6], though their overall goal was different from the reinforcement learning domain.

**Baseline** An ideal baseline would be any point along the decision boundary, where predictions are neutral. However, as showed by the work performed by [25], when the reference baseline is fixed to a specific value, attribution methods are not affine invariant. Instead, we define baseline as:

$$baseline_x_i = \mathbb{E}_{x_i} [\mathbb{E}_{y}[y|do(x_i = \alpha)]]$$

Rationally, the baseline is defined as $\mathbb{E}_{y}[y|do(x_i = \hat{x_i})]$, which includes all values of the input feature. If the expected value $y$ is constant for all possible intervention values $\alpha$, the baseline would also be the same constant and result in the causal attribution $ACE_{do(x_i = \alpha)}^{y} = 0$. Furthermore, the L1-norm of the causal attribution – $\mathbb{E}_{x_i} [||\mathbb{E}_{y}[y|do(x_i = \alpha)]||]$ could serve as a metric of average importance of a given attribute. This importance metric is then normalized for easier interpretation. Note that the proposed framework is model-agnostic and can be adapted to similar models which work with latents (TE-PPO and ATE-PPO in this paper). All the causal attributions computed in this paper have been over the agent trained on ATE-PPO. Although the approximation is very coarse, we can still obtain various insights from the proposed framework.
B Variational Bound Inference.

We borrow ideas from variational inference literature to introduce an information-theoretical regularization that encourages versatile skills. In particular, we present a lower entropy of marginal entropy $H[p(x)]$, which will prove helpful when applied to our objective function from Sec. 4.

Proposition B.1. The lower bound on the marginal entropy $H[p(x)]$ corresponds to:

\[
H[p(x)] \geq \int \int p(x, z) \log \left( \frac{q(z|x)}{p(x, z)} \right) dz \, dx,
\]

where $q(z|x)$ is the variational posterior

Proof.

\[
H[p(x)] = \int -p(x) \log[p(x)] \, dx = \int p(x) \log \left( \int q(z|x) \frac{1}{p(x)} \, dz \right) \, dx
\]

\[
= \int p(x) \log \left( q(z|x) \frac{p(z|x)}{p(x, z)} \right) \, dx \geq \int p(x) \int p(z|x) \log \left( \frac{q(z|x)}{p(x, z)} \right) \, dz \, dx
\]

\[
= \int \int p(x, z) \log \left( \frac{q(z|x)}{p(x, z)} \right) \, dx \, dz
\]

From the above Equation 11, we can construct a lower bound for our entropy term $H[\pi(a|s, t)]$ as follows:

\[
H[\pi(a|s, t)] \geq \mathbb{E}_{\pi_\phi(a, z|s, t)} \left[ \log \left( \frac{q(z|a, s, t)}{\pi(a, z|s, t)} \right) \right]
\]

\[
= \int \int p(\pi_\phi(a, z|s, t)) \log \left( \frac{q(z|a, s, t)}{\pi(a, z|s, t)} \right) \, da \, dz
\]

\[
= \int \int p(z|a, s, t) \pi(a|s, t) \log \left( \frac{q(z|a, s, t)}{\pi(a, z|s, t)} \right) \, da \, dz
\]

\[
= \int \int p(z|a, s, t) \pi(a|s, t) \left[ \log (q(z|a, s, t)) - \log (\pi(a, z|s, t)) \right] \, da \, dz
\]

\[
= \int \int \pi(a|s, t) p(z|a, s, t) \log (q(z|a, s, t)) \, da \, dz
\]

\[
- \int \int \pi(a|s, t) p(z|a, s, t) \log (\pi(a, z|s, t)) \, da \, dz
\]

\[
= - \int \int \pi(a|s, t) \mathcal{CE}[p(z|a, s, t)|q(z|a, s, t)] \, da
\]

\[
- \int \int \pi(a|s, t) p(z|a, s, t) \log (\pi(a, z|s, t)) \, da \, dz
\]

\[
= - \mathbb{E}_{\pi(a|s, t)} \left[ \mathcal{CE}[p(z|a, s, t)|q(z|a, s, t)] \right]
\]

\[
- \int \int \pi(a|s, t) p(z|a, s, t) \log (\pi(a, z|s, t)) \, da \, dz
\]
where $CE$ is cross entropy. To simplify the second part of the result from Equation 12, we simplify as follows:

\[
- \int \int \pi(a|s,t)p(z|a,s,t) \log (\pi(a,z|s,t)) \, da \, dz - \int \int p(z|a,s,t) \log (p(z|s,t)) \, da \, dz
\]

\[
= - \int \int \pi(a|s,t)p(z|a,s,t) \log (\pi(a,z|s,t)) \, da \, dz - \int \int \pi(a|s,t)p(z|a,s,t) \log (\pi(a|s,t,z)) \, da \, dz
\]

\[
= - \int \int p(z,a|s,t) \log (p(z|s,t)) \, da \, dz - \int \int \pi(a|s,t)p(z|a,s,t) \log (\pi(a|s,t,z)) \, da \, dz
\]

\[
= - \int \int p(z|s,t) \log (p(z|s,t)) \, dz - \int \int \pi(a|s,t)p(z|a,s,t) \log (\pi(a|s,t,z)) \, da \, dz
\]

(13)

Since the skill embedding is conditionally independent of the state of the agent given task $t$, we can simplify $p(z|s,t)$ as $p(z|t)$. Similarly, since the action $a$ is conditionally independent of the task given the latent $z$, we can simplify $\pi(a|s,t,z)$ as $\pi(a|s,z)$. This is possible since $z$ carries all the necessary information from $t$ which is required to solve the task. With the above simplifications, Equation 13 further simplifies to:

\[
= - \int p(z|s,t) \log (p(z|s,t)) \, dz - \int \int p(z|s,t)\pi(a|s,t,z) \log (\pi(a|s,t,z)) \, da \, dz
\]

\[
= \mathcal{H}[p(z|t)] + \int p(z|t)\mathcal{H}[\pi(a|s,z)] \, dz
\]

\[
= \mathcal{H}[p(z|t)] + \mathbb{E}_{p(z|t)} [\mathcal{H}[\pi(a|s,z)]]
\]

(14)

Note that $q(z|a,s,t)$ is the variational inference distribution that we are free to choose. Since $q(z|a,s,t)$ is intractable, we resort to a sample based evaluation of the Cross Entropy term. This bound holds for any $q$. Similar to [21], we avoid conditioning $q$ on task $t$ to ensure that a given trajectory alone will allow us to identify its skill embedding. Substituting the results from Equation 13, Equation 14, we can rewrite Equation 12 as:

\[
\mathcal{H}[\pi(a|s,t)] = - \mathbb{E}_{\pi(a|s,t)} [CE[p(z|a,s,t)||q(z|a,s,t)]] + \mathcal{H}[p(z|t)] + \mathbb{E}_{p(z|t)} [\mathcal{H}[\pi(a|s,z)]]
\]

\[
= - \int \pi(a,s,t)p(z|a,s,t) \log [q(z|a,s)] \, da + \mathcal{H}[p(z|t)] + \mathbb{E}_{p(z|t)} [\mathcal{H}[\pi(a|s,z)]]
\]

\[
= \int p(a,z|s,t) \log [q(z|a,s)] \, da + \mathcal{H}[p(z|t)] + \mathbb{E}_{p(z|t)} [\mathcal{H}[\pi(a|s,z)]]
\]

\[
= \mathbb{E}_{p(a,z|s,t)} [\log [q(z|a,s)]] + \mathcal{H}[p(z|t)] + \mathbb{E}_{p(z|t)} [\mathcal{H}[\pi(a|s,z)]]
\]

(15)

C Theoretical Results

We will first prove that the minimax game has a global optimum for a given policy network and a given embedding network in the case of an on-policy setting, where the optimal policy is already known to us. This proof is presented in Appendix C.1.

These proofs guarantee the working of our algorithm and set the necessary conditions for the model to train appropriately.
C.1 On-Policy Optimality

Proposition C.1. For a given $E$, the optimal policy $\pi$ is

$$\pi^*_E(t) = \arg\max_{\pi} Q_{\pi}^E(s, a, z, t)$$

and, the bound on $Q_{\pi}^E(s, a, z, t)$ is going to be such that:

$$Q_{\pi}^E(s, a, z, t) \leq \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma}$$

Proof. The training criterion for the given policy network $\pi$, given any encoder $E$, is to maximize the quantity $V(E, \pi)$ from Equation 3. Since the encoder $E$ is kept fixed, the last term is going to be fixed, and we can omit the term. Our equation now reduces to:

$$V(E, \pi) = \max_{\pi} Q_{\pi}^E(s, a, z, t)$$

$$= \max_{\pi} \sum_{i=0}^{\infty} \gamma^i \left[ r_i(s_i, a_i) + \alpha_2 \log \|q(z|a_i, s_i^H)\| + \alpha_3 H(\pi(a|s, z)) \right]$$

Here, we will assume the rewards to be bounded by the range $[0, R_{\max}]$. Note that any reward function, sparse or otherwise can be transformed to the above range, and can be subject to the same proof that follows. Furthermore, we assume perfect optimality of the inference network $q$ and the resulting Cross Entropy loss can be omitted since it would be 0. With the use of theory from Kullback–Leibler divergence and Shannon Entropy, we know that $H(x) \leq \log |x|$. In our case, the entropy $H(\pi(a|s, z))$ can be bounded to $\log |a_t|$, where $a_t$ is the action space of the given task. For simplification, we approximate $|a_t|$ to $|a_{\max}|$, where $a_{\max}$ denotes the action space with maximum cardinality. We can further simplify the equation above as follows:

$$Q_{\pi}^E(s, a, z, t) = \sum_{i=0}^{\infty} \gamma^i \left[ r_i(s_i, a_i) + \alpha_2 \log \|q(z|a_i, s_i^H)\| + \alpha_3 H(\pi(a|s, z)) \right]$$

$$\leq \sum_{i=0}^{\infty} \gamma^i \left[ R_{\max} + \alpha_3 \log |a_{\max}| \right]$$

$$\leq \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma}$$

The Equation 3 can now be reformulated as:

$$C(E) = \min_{E} V(E, \pi^*)$$

$$= \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma} - \alpha JSD(z, t)$$

(18)

Proposition C.2. The global minimum of the training criterion $C(E)$ is achieved if and only if

$$\alpha > \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma}$$

(19)

At this point, $C(E)$ achieves a minimum value bounded by $C(E) < 0$ and is not a trivial solution where the embedding function is an identity function. Furthermore, the property that $z$ and $t$ are mutually exclusive is held.

Proof. Since the Jensen–Shannon divergence between two distributions is always non-negative and zero if they are equal. Since we want to minimize Equation 18, we escape the trivial solution where the embedding network is an identity function. Note that, since the input to the embedding network is a one-hot encoded embedding of the task id, it satisfies the condition of the sharp peak and is uniform in terms of tasks and skills. For the time being, let us assume $\alpha$ is high enough and the $JSD$ objective is being tuned, rather than the term that helps deceive the agent network. In this scenario, we
have two unique cases: (i) when \( z \sim t \), and when (ii) \( z \not\sim t \). Note that the Jensen-Shannon divergence would be equal to 0 in the first case and positive in the second. Since our goal is to minimize the objective, the model would converge towards the goal where \( z \not\sim t \), and escape the trivial solution of the identity function. However, to ensure that the \( JS\alpha \) objective is being optimized with higher precedence, we set \( \alpha \) such that:

\[
\alpha JS\alpha(z, t) > \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma} \tag{20}
\]

Since we have already shown that the optimal solution is when \( z \not\sim t \), or when \( z \) and \( t \) are mutually exclusive, and JSD would be at its maximum value of 1. This mutual exclusivity property is essential since we would like the same skill to be shared across tasks, as long as the tasks are based on similar task structures. For instance, turning a doorknob or screwing a cap on a bottle involves identical skills. Using these fundamental skills to improve task structure in meta-learning would facilitate more accessible posterior adaptation and causal inference. With the assurance that our intuition is following the mathematical rigor, we can select \( \alpha \) such that:

\[
\alpha > \frac{R_{\max}}{1 - \gamma} + \alpha_3 \frac{\log |a_{\max}|}{1 - \gamma} \tag{21}
\]

Provided this condition is met, the training criterion \( C(E) \) achieves a minimum value bounded such that \( C(E) < 0 \). We select hyperparameters \( (\alpha_3, \alpha) \) such that the following condition is satisfied.

**D Tractable optimization of objective**

Here, we show that the equation presented in Equation 3 might not be the ideal form to compute the loss. This is because the computation of the Jensen-Shannon Divergence forces the shape of both skill \( z \) and task \( t \) to be the same. Furthermore, since \( z \) is derived from \( t \), it is not quite straightforward how to compute \( H(t|z) \). To subvert these issues, we simplify the above equation as follows.

Recall,

\[
H(t) - H(t|z) = H(z) - H(z|t) \tag{22}
\]

Note that our original objective equation does have the L.H.S. term. Thus, we simplify the objective to:

We simplify \( H(t|z) \) as \( H(t) - H(z) + H(z|t) \). Note that the \( H(t) \) term cancels out and we are left with \( 2\alpha(H(z) - H(z|t)) \) as our second term in Equation 3. This simplification is what we use while computing, but use the Jensen-Shannon Divergence simplification from Equation 3 to complete our proof of convergence of the model.

\[
O_V(E, \pi) = [Q^\pi_\nu(s, a; z, t)] - \alpha \mathbb{E}_{t \in T} [H(z) - H(z|t) + H(t) - H(t|z)]
= [Q^\pi_\nu(s, a; z, t)] - \alpha \mathbb{E}_{t \in T} [H(z) - H(z|t) + H(z) - H(z|t)] \tag{23}
= [Q^\pi_\nu(s, a; z, t)] - 2\alpha \mathbb{E}_{t \in T} [H(z) - H(z|t)]
\]

Since \( \alpha \) is an hyperparameter of our choice, we can assign \( 2\alpha \) as \( \alpha' \), and simplify the equation as follows:

\[
O_V(E, \pi) = [Q^\pi_\nu(s, a; z, t)] - \alpha' \mathbb{E}_{t \in T} [H(z) - H(z|t)] \tag{24}
\]

The above equation is what we use to make the computation tractable.

**E Limitations of Adversarial Training Regime**

Working in an adversarial training regime does pose its own set of limitations as discussed briefly in this section. If the hyperparameters are not carefully set, it might become unstable and inconsistent when the learning from the protagonist are completely nullified by the adversary. This could take place if the conditions highlighted in Appendix C are not met. This could also happen if the number.
of iterations the adversary is trained is much greater than the number of iterations of the protagonist forcing the final weights to be closer to an agent trying to minimize the rewards instead of maximize. Furthermore, setting $\alpha$ in Equation 1 very low will give more importance to learning skills that fool the agent but not necessarily diverse and distinct. Although our adversarial framework does outperform the traditional training regime on multiple environments, it is difficult to support the need for such a framework due to its inherent counter-intuitiveness, and requires more theoretical studies to understand this behavior.

### F Embedding Efficiency

We want our learned skills to be efficient and diverse in the latent embedding space. To evaluate the efficiency of our embeddings, we compute the diversity of these latents as the volume parallelopiped. Let us suppose our embedding vector is $z \sim R^n$. If the volume the skills encompass is higher, the more volume of the $R^n$ space it covers is higher, and we can better adapt to hierarchical skills with the same embedding space. With the given intuition, we compute the efficiency of the embedding as the square of the volume parallelopiped by the embedding. Appendix F.1 discusses our approach of computing this efficiency in more detail. Our experiments show that the latent skills learned from the ATE-PPO algorithm are more efficient and diverse than the latent skills learned from the TE-PPO algorithm. We summarize our findings in Table 2. Furthermore, when computing the volume of the space it covers, we can scale each latent feature by a constant, if needed, without any loss of information.

### F.1 Computing Volume of latent embedding

**Proposition F.1.** Let $\Pi$ be an $m$-dimensional parallelotope defined by edge vectors $B = \{v_1, v_2, ..., v_m\}$, where $v_i \in R^n$ for $n \geq m$. That is, we are looking at an $m$-dimensional parallelotope embedded inside $n$-dimensional space. Suppose $A$ is the $m \times n$ matrix with row vectors $B$ given by:

$$A = \begin{pmatrix}
      v_1^T \\
      \vdots \\
      v_m^T
    \end{pmatrix}$$

Then the $m$-dimensional volume of the parallelotope is given by:

$$[\text{vol}(\pi)]^2 = \det(AA^T)$$

**Proof.** Note that $AA^T$ is an $m \times m$ square matrix. Suppose that $m = 1$, then:

$$\det(AA^T) = \det(v_1v_1^T) = v_1 \cdot v_1 = \|v_1\|^2 = [\text{vol}(v_1)]^2$$

so the proposition holds for $m = 1$. From this base equation, we prove the above theorem by induction. Now, we induct on $m$.

Let us assume the proposition holds for $m'$ such $m' \geq 1$. If we can also prove that the proposition holds for $m' + 1$, we would have proved the above theorem. Letting $A_{m'}$ denote the matrix containing rows $v_1$ to $v_{m'}$, we can write $A = A_{m' + 1}$ as:

$$A = \begin{pmatrix}
      A_{m'} \\
      v_{m' + 1}
    \end{pmatrix}$$

We may decompose $v_{m' + 1}$ orthogonally as:

$$v_{m' + 1} = v_\perp + v_\parallel$$
where \( v_{\perp} \) lies in the orthogonally complement of the base (i.e., the height of our parallelepiped), and 
\[ v_{\perp} \cdot v_i = 0, \forall 1 \leq i \leq m. \]  
Furthermore, \( v_{\parallel} \) must be in the span of vectors \( \{v_1, v_2, ..., v_{m'}\} \), such that:

\[ v_{\parallel} = c_1 v_1 + ... c_{m'} v_{m'} \]

We apply a sequence of elementary row operations to \( A \), adding a multiple \(-c_i\) of row \( i \) to row \( m' + 1 \), \( \forall 1 \leq i \leq m' \). We can then write the resulting matrix \( B \) as:

\[
B = \begin{pmatrix}
A_{m'}
\end{pmatrix}
\begin{pmatrix}
v_{\perp}^T
\end{pmatrix} = E_{m'}...E_1 A,
\]

where each \( E_i \) is an elementary matrix adding a multiple of one row to another. Notice that the above operation corresponds to shearing the parallelotope so that the last edge is perpendicular to the base.

We see that these operations do not change the determinant as:

\[
\det(BB^T) = \det(E_{m'}...E_1 (AA^T) E_1^T ... E_{m'}^T) = \det(AA^T)
\]

Through block multiplication, we can obtain \( BB^T \) as follows:

\[
BB^T = \begin{pmatrix}
A_{m'}
\end{pmatrix}
\begin{pmatrix}
v_{\perp}^T
\end{pmatrix} = E_{m'}...E_1 A
\]

Furthermore, notice that

\[
A_{m'}v_{\perp} = \begin{pmatrix}
v_{\perp}^T
\end{pmatrix} v_{\perp} = 0
\]

Therefore, we have

\[
BB^T = \begin{pmatrix}
A_{m'}A_{m'}^T
\end{pmatrix}
\begin{pmatrix}
\begin{pmatrix}
0
\end{pmatrix}
\end{pmatrix}
\]

Taking the determinant, we can simplify \( \det(BB^T) \) as

\[
\det(BB^T) = \|v_{\perp}\|^2 \det(A_{m'}A_{m'}^T)
\]

By definition, \( \|v_{\perp}\| \) is the height of the parallelotope, and by the induction hypothesis, \( \det(A_{m'}A_{m'}^T) \) is the square of the base. Therefore, we have proved the above theorem by induction.

\[ \square \]

### G Analysis of Skills learned

In this section, we discuss a few additional results, along with a study of the effect of each latent feature on the agent’s behavior.

#### G.1 PointMass Environment

From the ACE plots, we can successfully assert that feature \( z_2 \) is an auxiliary variable and does not play any significant role in the model’s behavior since the ACE value is close to 0. Furthermore, we notice that the \( z_3 \) and \( z_4 \) causal analysis are pretty similar, and \( z_1 \) seems to be an essential feature in the pool of latent features.

To better understand the effect of each feature on the behavior of the model, we use an input perturbation method. Here we fix all features to a given task: say Top, but vary a given feature within the skill to study the change in the agent’s behavior. Figure 10 depicts the results from our input perturbation experiment.
Figure 5: The Average Causal Effect of the latent skills for different tasks in the PointMass Environment. From the above plots, we can make a few inferences, such as the fact that embedding $z_2$ seems to be an auxiliary feature and does not play a significant role in the behavior of the agent.

G.2 2-D Navigation Environment

Figure 6: The Average Causal Effect of the latent skills for different tasks in the 2-D Navigation Environment. From the above plots, we notice that there is a lot of overlap and noise in the ACE, especially in the case of Figure 6(b) and Figure 6(c). This is expected since the model needs to learn to diverge from its initial path after a few time steps, bringing exploration only after it crosses the corridor. We hypothesize that we do not observe the same noise in Figure 6(a) since there is no divergence from the path.

The average importance of each feature is shown in Figure 7.

Figure 7: The normalized importance of each latent feature on the 2-D navigation tasks across 5 different seeds.

From the ACE plots, we can successfully assert that feature $z_2$ is an auxiliary variable and does not play any significant role in the model’s behavior since the ACE value is close to 0. Furthermore, we notice that the $z_3$ and $z_4$ causal analysis are pretty similar, and $z_1$ seems to be an essential feature in the pool of latent features.

To better understand the effect of each feature on the behavior of the model, we use an input perturbation method. Here we fix all features to a given task: say Top, but vary a given feature within the skill to study the change in the agent’s behavior. Figure 11 depicts the results from our input perturbation experiment.
Figure 8: The Average Causal Effect of the latent skills for different tasks in the Metaworld (MT5) Environment. From the above plots, we notice that there is a lot of overlap and noise in the ACE. Furthermore, we notice that the ACE of the tasks: Push and Drawer Open, is roughly zero across all latents, and our model has prioritized learning other tasks.

G.3 Meta-World(MT5) Environment

The average importance of each feature is shown in Figure 9.

Figure 9: The normalized importance of each latent feature on the Metaworld (MT5) tasks across 5 different seeds.

From the ACE plots, we can successfully assert that feature $z_2$ is an auxiliary variable and does not play any significant role in the model’s behavior since the ACE value is close to 0. Furthermore, we notice that the $z_1$ variable is significant for tasks that involve closing, whereas $z_2$ and $z_3$ variable are significant when tasks involve the action of opening.

Performing an input-perturbation method, similar to previous sections might not be feasible for the MetaWorld environment, as it involves much higher complexity of observation and action space for a simple visualization.
(a) The effect of $z_1$ on the behavior of the environment. As we can notice from the above plots, $z_1$ seems to have a significant effect on the behavior of the agent. We hypothesize that the $z_1$ feature is used to determine the top-left and bottom-right segment of tasks. This would explain why some trajectories, specifically Top/Left and Bottom/Right are similar to each other.

(b) The effect of $z_2$ on the behavior of the environment. As we can notice from the above plots, $z_2$ seems to have little to no effect on the behavior of the agent. We hypothesize that the $z_2$ feature could have low importance and unnecessary, or could be a basic skill used across all tasks. This would explain why perturbation of the $z_2$ feature does not lead to any difference in behavior, i.e. not a discriminating feature across tasks.

(c) The effect of $z_3$ on the behavior of the environment. As we can notice from the above plots, $z_3$ seems to have a significant effect on the behavior of the agent. We hypothesize that the $z_3$ feature is used to determine the top-right and bottom-left segment of tasks. This would explain why some trajectories, specifically Top/Right and Bottom/Left are similar to each other.

(d) The effect of $z_4$ on the behavior of the environment. As we can notice from the above plots, $z_4$ seems to have a similar purpose as the $z_3$ feature. However, we notice a slight difference in the trajectory - $z_4$ enforces a shorter distance before the sudden turn in Left/Right, but a longer one in Top/Bottom. Whereas, $z_3$ does the exact opposite.

Figure 10: Input perturbation results on point mass environment
(a) The effect of $z_1$ on the behavior of the environment. As we can notice from the above plots, $z_1$ does not seem to have a significant effect on the behavior of the agent. We hypothesize that the $z_1$ feature could have low importance and unnecessary, or could be a basic skill used across all tasks. This would explain why perturbation of the $z_1$ feature does not lead to any difference in behavior, i.e. not a discriminating feature across tasks.

(b) The effect of $z_2$ on the behavior of the environment. As we can notice from the above plots, $z_2$ seems to have little to no effect on the behavior of the agent. This feature seems to play a very similar role to $z_1$ as discussed earlier.

(c) The effect of $z_3$ on the behavior of the environment. As we can notice from the above plots, $z_3$ seems to have a significant effect on the behavior of the agent. We hypothesize that the $z_3$ feature helps decide the behavior of the agent for Top and Right goals. However, when it comes to the Bottom task, a feature other than $z_3$ overwrites this behavior, and forces the trajectory to go towards the Bottom goal.

(d) The effect of $z_4$ on the behavior of the environment. As we can notice from the above plots, $z_4$ seems to be the only feature that helps propel the trajectory towards the Bottom goal. This is the feature that overwrites the effect of $z_3$ feature and forces the action to be towards the destined goal.

Figure 11: Input perturbation results on 2-D Navigation environment
## H Model and Hyperparameters

This section discusses the model and hyperparameters used in our experiments.

Table 3 depicts the hyperparameters we used for training our TE-PPO and ATE-PPO algorithm on the point mass environment.

Similarly, Table 4 depicts the hyperparameters we used for training our TE-PPO and ATE-PPO algorithm on the 2-D navigation environment.

Finally, Table 5 depicts the hyperparameters we used for training our TE-PPO and ATE-PPO algorithm on the Meta-World (MT5) environment.

| Description                  | TE-PPO | ATE-PPO | argument_name         |
|------------------------------|--------|---------|-----------------------|
| **General Hyperparameters**  |        |         |                       |
| Discount                     | 0.99   | 0.99    | discount              |
| Batch size                   | 4096   | 4096    | batch_size            |
| Number of epochs             | 600    | 600     | n_epochs              |
| **Algorithm-Specific Hyperparameters** |   |   |                       |
| Encoder hidden sizes         | (20, 20) | (20, 20) | enc_hidden_sizes      |
| Inference hidden sizes       | (20, 20) | (20, 20) | inf_hidden_sizes      |
| Policy hidden sizes          | (32, 16) | (32, 16) | pol_hidden_sizes      |
| Activation function of hidden layers | tanh | tanh | hidden_nonlinearity |
| Likelihood ratio clip range  | 0.2    | 0.2     | lr_clip_range         |
| Latent dimension             | 2      | 4       | latent_length         |
| Inference window length      | 6      | 6       | inference_window      |
| Embedding maximum standard deviation | 0.2 | 0.2 | embedding_max_std    |
| Policy entropy coefficient   | $1e^{-3}$ | $1e^{-3}$ | policy_ent_coeff      |
| Encoder entropy coefficient  | $1e^{-3}$ | $1e^{-3}$ | enc_ent_coeff         |
| Inference entropy coefficient| $5e^{-2}$ | $5e^{-2}$ | inf_ent_coeff         |
| **Optimizer-Specific Hyperparameters** | | |                       |
| Protagonist mini-batch size  | 32     | 64      | pr_batch_size         |
| Adversary mini-batch size    |         | 64      | ad_batch_size         |
| Inference mini-batch size    | 32     | 64      | inf_batch_size        |
| Protagonist learning rate    | $1e^{-4}$ | $1e^{-3}$ | pr_lr                 |
| Adversary learning rate      |         | $1e^{-4}$ | ad_lr                 |
| Inference learning rate      | $1e^{-3}$ | $1e^{-3}$ | inf_lr                |

Table 3: Hyperparameters used for training TE-PPO and ATE-PPO on pointmass environment.
| Description                  | TE-PPO | ATE-PPO | argument_name |
|------------------------------|--------|---------|---------------|
| **General Hyperparameters**  |        |         |               |
| Discount                     | 0.99   | 0.99    | discount      |
| Batch size                   | 3072   | 3072    | batch_size    |
| Number of epochs             | 400    | 400     | n_epochs      |
| **Algorithm-Specific Hyperparameters** |        |         |               |
| Encoder hidden sizes         | (20, 20) | (20, 20) | enc_hidden_sizes |
| Inference hidden sizes       | (20, 20) | (20, 20) | inf_hidden_sizes |
| Policy hidden sizes          | (32, 16) | (32, 16) | pol_hidden_sizes |
| Activation function of hidden layers | tanh    | tanh    | hidden_nonlinearity |
| Likelihood ratio clip range  | 0.2    | 0.2     | lr_clip_range |
| Latent dimension             | 4      | 4       | latent_length |
| Inference window length      | 6      | 6       | inference_window |
| Embedding maximum standard deviation | 0.2    | 0.2     | embedding_max_std |
| Policy entropy coefficient   | 1e^{-3} | 1e^{-3} | policy_ent_coeff |
| Encoder entropy coefficient  | 1e^{-3} | 1e^{-3} | enc_ent_coeff |
| Inference entropy coefficient| 5e^{-2} | 5e^{-2} | inf_ent_coeff |
| **Optimizer-Specific Hyperparameters** |        |         |               |
| Protaganist mini-batch size  | 32     | 64      | pr_batch_size |
| Adversary mini-batch size    | −      | 32      | ad_batch_size |
| Inference mini-batch size    | 32     | 64      | inf_batch_size |
| Protaganist learning rate    | 1e^{-4} | 5e^{-4} | pr_lr |
| Adversary learning rate      | −      | 1e^{-4} | ad_lr |
| Inference learning rate      | 1e^{-3} | 5e^{-4} | inf_lr |

Table 4: Hyperparameters used for training TE-PPO and ATE-PPO on 2-D Navigation environment.
## Hyperparameters for Training TE-PPO and ATE-PPO on MT5 Environment

| Description                  | TE-PPO | ATE-PPO | argument_name          |
|------------------------------|--------|---------|------------------------|
| General Hyperparameters      |        |         |                        |
| **Discount**                 | 0.99   | 0.99    | discount               |
| **Batch size**               | 25000  | 25000   | batch_size             |
| **Number of epochs**         | 1000   | 1000    | n_epochs               |
| Algorithm-Specific Hyperparameters |       |         |                        |
| **Encoder hidden sizes**     | (20, 20) | (20, 20) | enc_hidden_sizes       |
| **Inference hidden sizes**   | (20, 20) | (20, 20) | inf_hidden_sizes       |
| **Policy hidden sizes**      | (32, 16) | (32, 16) | pol_hidden_sizes       |
| **Activation function of hidden layers** | tanh | tanh | hidden_nonlinearity   |
| **Likelihood ratio clip range** | 0.2  | 0.2  | lr_clip_range          |
| **Latent dimension**         | 4      | 4      | latent_length          |
| **Inference window length**  | 6      | 6      | inference_window       |
| **Embedding maximum standard deviation** | 0.2  | 0.2  | embedding_max_std     |
| **Policy entropy coefficient** | $2e^{-2}$ | 2$e^{-2}$ | policy_ent_coeff      |
| **Encoder entropy coefficient** | $2e^{-2}$ | 2$e^{-2}$ | enc_ent_coeff        |
| **Inference entropy coefficient** | $5e^{-2}$ | 5$e^{-2}$ | inf_ent_coeff       |
| Optimizer-Specific Hyperparameters |       |         |                        |
| **Protaganist mini-batch size** | 256  | 256    | pr_batch_size          |
| **Adversary mini-batch size** | −      | 256    | ad_batch_size          |
| **Inference mini-batch size** | 256    | 256    | inf_batch_size         |
| **Protaganist learning rate** | $1e^{-3}$ | 5$e^{-4}$ | pr_lr                 |
| **Adversary learning rate**  | −      | $1e^{-4}$ | ad_lr                 |
| **Inference learning rate**  | $1e^{-3}$ | 5$e^{-4}$ | inf_lr                |

Table 5: Hyperparameters use for training TE-PPO and ATE-PPO on MT5 environment.