Overcoming State and Action Space Disparities in Multi-Domain, Multi-Task Reinforcement Learning

Reginald McLean
Toronto Metropolitan University
reginald.mclean@torontomu.ca

Kai Yuan
Intel Corporation

Isaac Woungang
Toronto Metropolitan University

Nariman Farsad
Toronto Metropolitan University

Pablo Samuel Castro
Université de Montréal, Mila

Abstract: Current multi-task reinforcement learning (MTRL) methods have the ability to perform a large number of tasks with a single policy. However when attempting to interact with a new domain, the MTRL agent would need to be re-trained due to differences in domain dynamics and structure. Because of these limitations, we are forced to train multiple policies even though tasks may have shared dynamics, leading to needing more samples and is thus sample inefficient. In this work, we explore the ability of MTRL agents to learn in various domains with various dynamics by simultaneously learning in multiple domains, without the need to fine-tune extra policies. In doing so we find that a MTRL agent trained in multiple domains induces an increase in sample efficiency of up to 70% while maintaining the overall success rate of the MTRL agent. Code for reproducing experiments is available on GitHub.

Keywords: Multi-Task Reinforcement Learning, Morphology-Agnostic Policies

1 Introduction

Reinforcement learning (RL) has shown remarkable success in its application to many different scenarios including game playing [1], controlling stratospheric balloons [2], and controlling fusion reactors [3]. In order to scale RL to accomplish multiple tasks with a single policy, multi-task RL learns across multiple tasks which may accelerate learning in additional tasks which improves sample efficiency. However, current online multi-task RL algorithms are unable to learn across tasks that have different state and action spaces, and different semantics.

To overcome this gap in the reuse of RL policies in tasks and domains, this paper aims to study the design decisions and training protocols required for training a single policy across multiple domains, and evaluates their performance in robotic manipulation domains in different domains. For a RL agent to have generalized skills, the RL agent must be able to leverage it's previous experiences in new tasks. An example of this would be a RL agent that can pick up a frying pan in a kitchen domain and can also pick up a remote control in a living room domain. Various approaches have been proposed to address the challenge of transferring skills across different domains and robot morphologies. Recent works have leveraged offline reinforcement learning where a large dataset of trajectories are used to enable effective learning on different robots [4][5][6][7][8]. However, these policies are trained using datasets of successful trajectories limiting their ability to acquire new skills. Alternatively, online RL approaches generally either receive joint specific observations [9] or attempt to infer robot morphology from the reinforcement learning objective [10]. We differ from these lines of work as we perform all learning in an online fashion while utilizing an architecture that can handle varying state and action spaces without requiring joint-specific observations or explicit morphology inference. Our approach, which we call SAL (**S**hared, domain-**A**gnostic, **L**atent space), enables effective skill transfer across different domains and robot morphologies in an online setting.

To address these challenges, SAL employs a unique architecture designed to overcome the limitations of varying state and action spaces. A central challenge impeding training a single policy across domains is the difference in state and action space sizes that prevents any direct reuse of a trained policy, as the policy can have different input and output sizes for each specific domain. This work aims to enable the transfer of skills between domains with different robotic morphologies, where both the dimensions and semantic meanings of state and action spaces differ. This domain-agnostic transfer is enabled by learning the appropriate latent representations through state and action translation layers, which map the states to a latent state space and then extract an action from latent space to an domain-specific action space, respectively. This architecture is an extension of the multi-headed architecture from [11] and [12]. We demonstrate the effectiveness of these layers in continuous control tasks that transfer high-level manipulation concepts between the domains. Our experiments show that learning a policy in a shared, domain-agnostic latent space by translating states to this SAL space and decoding latent skills into actions from the SAL space yields sample efficiency gains anywhere from 7% to 72% compared to training on a single domain's tasks alone.

The proposed method, and thus the efficient skill transfer across domains, can be achieved through minimal modifications to existing RL algorithms. We demonstrate skill transfer between the Meta-World [11] and Franka Kitchen [13] domains, and show how SAL can be applied to state of the art algorithms, such as Soft-Actor Critic (SAC) [14].

Our contributions are as follows:

- We propose the shared, domain-agnostic, latent policy architecture, which enables multi-task reinforcement learning across domains with different state and action spaces by using a single policy in Section 3.
- We show that learning with our proposed architecture can accelerate learning by up to 72% in Section 4.4.
- We provide insights into how the latent space of the SAL architecture is organized to better understand the limitations and future work associated with learning across differing state and action spaces in Section 4.6.
- We provide an open-source implementation to apply SAL on existing RL algorithms.

2 Problem Statement

Reinforcement learning is formulated using a Markov decision process (MDP) [15], where an MDP $\mathscr{M}$ is a tuple of (S, A, P, r, γ, p), where S is the state space, A is the action space, the probability transition function $P : S \times A \rightarrow [0, 1]^{|s|}$, $r : S \times A \rightarrow \mathbb{R}$ the reward function, γ in $[0, 1)$ is the discount factor, and p is the initial state distribution.

At each time step, the agent observes the state at time t, chooses an action sampled according to some policy $\pi : S \rightarrow A$ based on s_t, receives a reward for landing in-state s_{t+1}, and observes state s_{t+1}. The goal of the agent is to find a policy that maximizes expected returns for the current task $E[R(\tau)]$ where $R(\tau)$ is the sum of discounted rewards along the trajectory induced by following the policy π.

In the multitask reinforcement learning problem, a task distribution must be chosen where N tasks are sampled from a task set T according to $t \sim n(t)$. Each task can be viewed as having its own MDP: $(S, A, P_i, r_i, \gamma, p_i)$ where the state space, action space, and discount factors are held constant across tasks while the probability transition function, reward function, and initial state distribution are specific to each task t_i. The goal of the multitask reinforcement learning agent is to maximize the expected sum of discounted rewards across each task $E_{t\sim p(t)}[E_{\tau\sim\pi}[R(\tau)]]$ using policy $\pi_\eta(a|s,z)$, where z is the task identifier and η are learnable parameters of the policy.

In this paper, the problem we consider is extending multitask RL to multiple domains that don't share the same state and action spaces. Let E be a set of M domains, where each domain $m \in E$ has a set of T_m distinct tasks. Each task $t_m \in T_m$ in each domain m has an MDP $\mathscr{M}_t^{(m)}$ for each task t. Therefore, the MDP $\mathscr{M}_j^{(i)}$ is now comprised of the tuple $(S^{(i)}, A^{(i)}, P_j^{(i)}, r_j^{(i)}, \gamma, p_j^{(i)})$, where the elements in the tuple can change with respect to the domain.

The domain and the task can be jointly sampled from a joint distribution $m, t \sim n(m,t)$. Since the state space $S^{(m)}$ and action space $A^{(m)}$ are domain-dependent, traditional multitask reinforcement learning algorithms cannot be applied to share policy parameters π_η between tasks because the input state and output actions can vary in dimensions. In order to overcome this problem, we propose using translation layers to the input and action heads for the output of the policy. The input translation layer maps varied state representations to a common latent state space, while the output action heads decodes actions from this latent space back to domain-specific action spaces. This approach allows us to maintain a consistent internal representation for the policy, regardless of the specific domain it's operating in.

To aid the reader in understanding the difference between domains and tasks, we use the following definition of domains and tasks. In Meta-World, we leverage MT10 which contains 10 tasks available in Meta-World. Each task t in MT10 are distinct tasks for the MTRL agent to accomplish such as: opening a window, closing a drawer, grasping an object and moving it to a goal. Thus the domain is Meta-World, where the tasks are the individual tasks available in MT10. Similarly, for the Franka Kitchen domain, there are several tasks for the MTRL agent to accomplish such as slide a cabinet, open the microwave, grasp a kettle by the handle and move it to a goal location. Thus in the Franka Kitchen domain, there are several tasks available for the MTRL agent to accomplish. Additional information on the Meta-World and Franka Kitchen domains can be found in Appendix B.2 and B.3.

With the use of translation layers and action heads, our goals are two-fold. The first goal is to determine if we can demonstrate skills transfer across domains. The second goal is to find the best training approach to learn a single policy π that can act in all domains, in terms of mean success rate and sample complexity.

3 Enabling Skill Transfer

In this section, we introduce our method for enabling the sharing of parameters across differing state and action spaces. In order to train using this multi-task algorithm we must first overcome the issue of differing state and action spaces.

3.1 Translation Layers

In order to overcome the difference in state spaces between domains, we propose the use of translation layers. A translation layer f_{θ_i} is parameterized by parameters θ_i where i is the index of an domain. The translation layer f_{θ_i} receives the state-vector $S_t^{(i)}$ from domain i and encodes the state-vector into the shared latent space of the SAL architecture. This allows for the translation layer f_{θ_i} to process the state-vector for domain specific features before embedding the features into the shared latent space. By using these translation layers, we have overcome the issue of differing state spaces by using domain specific modules.

3.2 Action Heads

To overcome the difference in action spaces, we propose the use of a shared space to generate actions inspired by [8], and then decode these shared action representations via domain specific action heads. Once the translation layer f_{θ_i} processes the state-vector $S_t^{(i)}$ from domain i, the

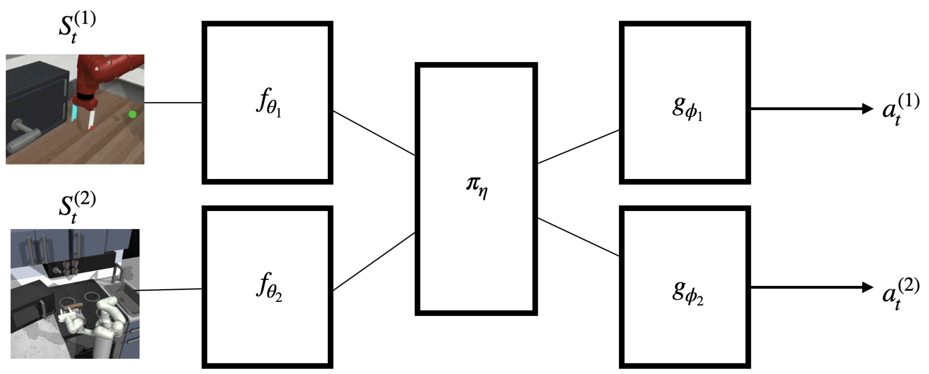


Figure 1: Control architecture for multi-task learning. The shared, domain-agnostic, latent (SAL) policy. To generate an action for a domain, the state vector from that domain is passed through the SAL network. First through translation layer f_{θ_i}, then shared policy layers π_η, and finally though domain specific action head a^i.

shared, domain-agnostic latent space $a_{sh} = \pi_\eta(f_{\theta_i}(S_t^{(i)}))$ encodes the features into a shared latent action space where a_{sh} is the output of the final parameters of the shared domain agnostic latent space π_η. Once the features are embedded into the shared action representation, the domain specific action heads decode this action into domain specific actions $g_{\phi_i}(a_{sh})$ where the action head i is parameterized by a set of parameters ϕ_i to generate action $a_t^{(i)}$.

3.3 Shared, domain-agnostic, latent (SAL) policy

To have an domain-agnostic policy that can learn in new domains while sharing parameters across diverse domains, we propose the SAL policy found in Figure 1. For some domain i to produce an action $a_t^{(i)}$ at time step t, a forward pass must be completed as $g_{\phi_i}\left(\pi_\eta\left(f_{\theta_i}(s_t^{(i)})\right)\right)$. These parameters are then optimized using the loss of the RL algorithm. In this paper we use Soft Actor Critic [14] for each domain but any RL algorithm can be used. The optimizer for each domain i is operating on the translation layer f_{θ_i}, the action head g_{ϕ_i}, and the shared policy parameters π_η. The SAL policy is designed and optimized in a way that shares parameters across diverse domains in the policy layers π_η allowing for relevant similarities across domains to be encoded in the Shared, domain-Agnostic, Latent policy space.

4 How to train a SAL policy

In order to learn the dynamics of multiple domains within a shared parameter space, we propose to use the SAL policy architecture. We use the domains of Meta-World [11] and Franka Kitchen [13]. Further information about these domains can be found in Appendix B.2 and B.3. The following subsections outline the process of searching for the best performance when using the SAL architecture and some of the current limitations of the method.

4.1 Issues when training on multiple domains

Due to the static nature of feed-forward neural network input and output dimensions, they cannot handle inputs or outputs of various sizes. One approach to overcome this issue would be to pad the state and action spaces with zeros to ensure that input and output dimensions are the same across domains similar to the approach in [4]. In these experiments we semantically align common state features, such as goal and task IDs, and pad missing dimensions with zeros. In order to overcome the issue of differing action spaces, we have our policy output actions the size of the largest action space, and then only use the appropriate action size for any domain that has a smaller action space. In Figure 2 we find that performance of a multi-task RL algorithm using this method is extremely

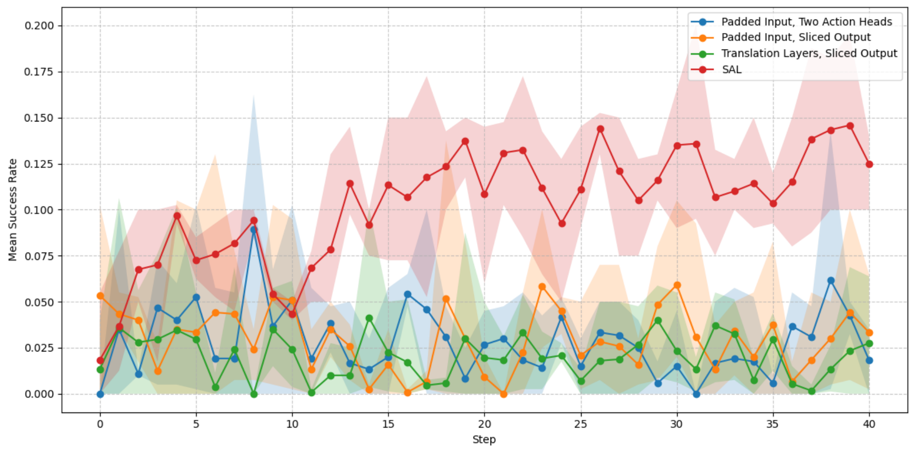


Figure 2: Mean success rate plots across all tasks from MT10 and Franka Kitchen. Blue plot is a network with padded inputs, and action heads. Orange plot is padded inputs, and outputting actions in the larger action space & only using needed action dimensions. Green plot uses translation layers and actions in the larger action space, like Orange. Red is the SAL architecture. Shaded regions indicate the minimum and maximum success rates. Success is determined by manipulating an object within the domain within Dcm of the goal, where D is determined by the task.

limited. In order to learn multiple sets of dynamics in a single policy network, we propose the use of SAL which aims to separate domain specific parameters into the translation layers and action heads, while sharing common features of domain dynamics in the shared parameters.

4.2 Training a SAL policy

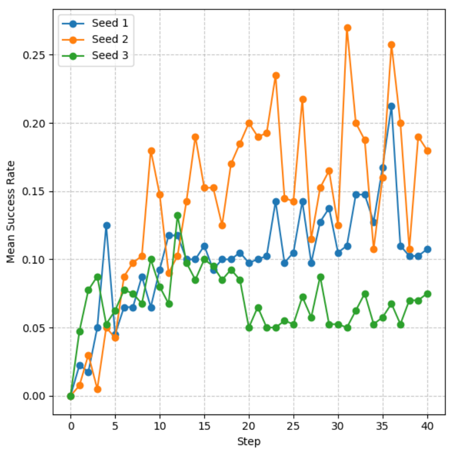
In Figure 2 we show the performance of SAL in reference to the padded state and action space baseline approaches. The difference in performance is about 10%, showing the benefit of extracting domain relevant features in the translation layers & action heads while also sharing some features in the shared parameters. One of the limitations of this current iteration of SAL is that it's difficult for the SAL shared parameters π_η to align task relevant features across domains. For example if task X in domain i is similar to task Y in domain j, there isn't a mechanism to enforce similarity between learned features for those tasks X and Y.

4.3 Task alignment

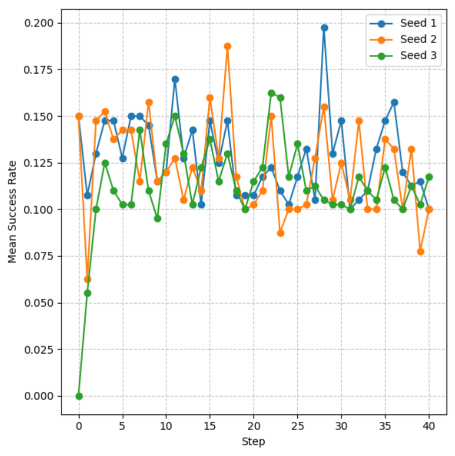
One approach to relate one task to another could be to do what we call task alignment on the task IDs that are input to the translation layers. This alignment process would enable re-use of policy parameters for similar tasks, leading to more sample efficient learning of similar tasks. To the best of the authors knowledge, there is no method for determining this alignment a priori to training a policy. Thus we explore random alignments of task IDs in order to see if there are any similar tasks that can be learned across domains. In Figure 3 we show the success rates of different alignments of tasks across multiple random seeds. We can see that there does seem to be some grouping in Figure 3a or Figure 3b, however it is not present for all tasks limiting the overall success rate of the agent when attempting to align all of the tasks present.

4.4 Manual task alignment

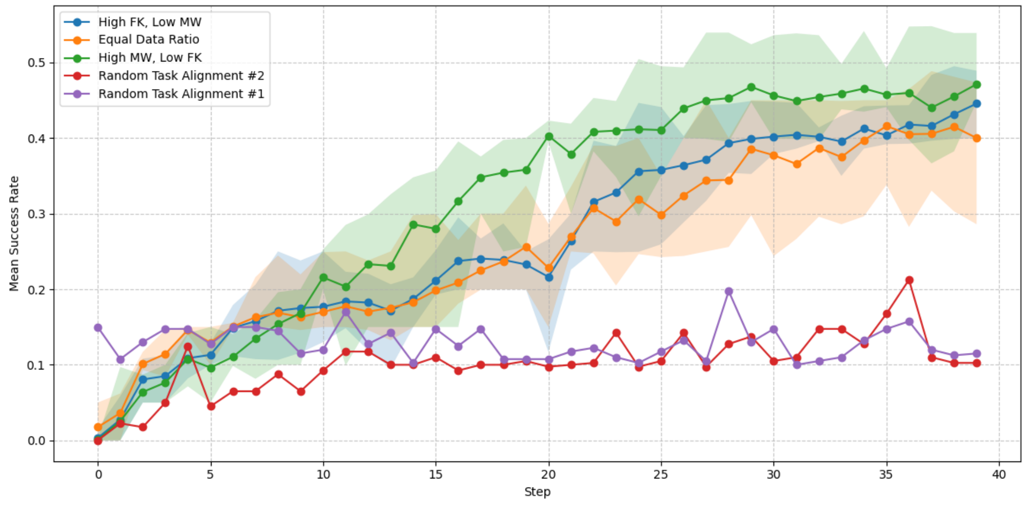
In order to better leverage the information gained from the random task alignments, we implement what we refer to as the manual task alignment. In this alignment technique we leverage the grouping information of the random task alignment experiments in Section 4.3, while also introducing some human bias towards similar tasks by manually choosing tasks that are similar across domains. For example, the pick place task in Meta-World is similar to the kettle task in Franka Kitchen as both



(a)



(b)



(c)

Figure 3: Figure 3a & 3b: success rates across 2 random task alignments, over 3 random seeds each. Additional seeds are included in Section 6. Figure 3c: mean success rate from the manual task alignment. Results indicate that SAL can learn an effective policy. Shaded regions indicate the minimum and maximum success rates. MW is Meta-World MT10 domain, and FK is Franka Kitchen domain. Random task alignments are included to highlight performance gain.

tasks require the agent to grasp and object and move it to a goal location. Therefore we align those task IDs to have the same value. For any tasks that do not have a similar task in another domain, we give these tasks unique task IDs. In Figure 3c we can see that by leveraging this manual task alignment method we have a substantial increase in success rate compared to the results found in Section 4.2 or 4.3. However, we also find that the sample complexity for learning tasks has worsened using this manual task alignment compared to learning the tasks individually. In Table 1 we find that by using the SAL architecture we increase the amount of samples the policy network needs to be trained on to learn tasks. While the manual task alignment shows promise in improving overall success rates, the increased sample complexity highlights a new challenge: balancing the learning across multiple domains efficiently. This trade-off between improved task generalization and increased sample requirements suggests that further refinement of our learning approach is necessary. To address this, we turn our attention to a more nuanced strategy for managing the learning process across multiple domains.

4.5 Data manipulation

While the task alignment approach in Section 4.4 showed some promise, its limitations in consistently grouping similar tasks across domains highlight the need for alternative strategies to enhance SAL's learning capabilities. One key challenge in multi-domain learning is balancing the exposure to different tasks to achieve efficient and robust learning. To address this, we propose a dynamic data sampling strategy that gradually shifts the policy's focus across domains. Initially, we create a limited data scenario where the policy uses 90% of the data from domain i and 10% of the data from domain j for its update. This imbalanced data ratio allows the policy to first build a strong foundation in one domain before gradually incorporating knowledge from the second. As training progresses, we slowly adjust this ratio towards a 50/50 split, enabling the policy to transfer and integrate knowledge across domains more effectively. This approach aims to build on the improvement of the manual task alignment in Section 4.4 by providing a more controlled and gradual exposure to multiple tasks. We hypothesize that SAL can develop more robust shared parameters while still maintaining domain-specific adaptations through its translation layers and action heads with this method.

Table 1 shows the results of two configurations of this data manipulation method, one where we use 90% Franka Kitchen data and 10% Meta-World data in the policy update and one where we use 90% Meta-World data and 10% Franka Kitchen data. These configurations are labeled High FK and High MW respectively. We find that the High FK induces more sample efficient learning across the Meta-World and Franka Kitchen tasks, while the High MW configuration induces some more sample efficient learning. These results validate the use of the SAL architecture and the data manipulation training configuration, as the goal of training simultaneously in these domains is to learn with a relatively high success rate while also learning different tasks in a more sample efficient manner. One of the limitations of these methods is the decrease in sample efficiency for non-aligned tasks. Table 4 and Table 5 show the number of samples needed to learn all tasks in either Meta-World or Franka Kitchen. For the un-aligned tasks, there is a decrease in sample efficiency.

4.6 SAL Latent Space Analysis

Finally, to examine the groupings of tasks made by training using the SAL policy architecture we plot the outputs of both the translation layers f_{θ_i} and the output of shared policy layers π_η. We use T-SNE embeddings [16] to visualize the latent vectors. In Figure 7 we show the T-SNE embeddings after the first epoch and after the last epoch. Figure 7c and Figure 7d show the outputs of the translation layers, while Figure 7a and Figure 7b show the outputs of the SAL policy layers. The SAL architecture shows the ability to group related tasks together, however it is still uncertain how to handle tasks that don't have any similar tasks in another domain.

5 Limitations and Future Work

We would like to highlight the limitations of the current approaches that were successful as they are important for both the future of this work, and important for the field of multi-task reinforcement learning.

5.1 Task Alignment

In Section 4.3 and Section 4.4 we outlined the methods that we used to align the task IDs of individual tasks across the various tasks from Meta-World and Franka Kitchen. However, these alignment methods have several limitations. As the number of tasks increases, manually aligning the tasks becomes more time-consuming and challenging. By allowing for a human to align the tasks, it's possible that the human will introduce biases that will limit the capabilities of the MTRL algorithm. The alignment method may not generalize well to new domains or the introduction of additional domains during training. As noted in Section 4.4, the manual alignment method without the data manipulation method increased the sample complexity, suggesting that there is a trade-off between generalization and learning efficiency. Thus these limitations highlight the need for further research into being able to determine how the actions from task i may transfer to task j.

5.2 Data Manipulation

In Section 4.5, we outline the method we use to learn in a more sample efficient manner where we slowly introduce data from one domain during a policy update. This method may have additional issues when transitioning to new domains. The strategy of using 90% data from one domain initially may lead to temporarily poor performance in the under-sampled domain. The performance of the system may be sensitive to how quickly or slowly the data ratio is adjusted, and finding the optimal schedule could be challenging. The optimal starting ratio and adjustment schedule might vary significantly between different sets of domains, potentially requiring extensive tuning. Our current implementation focuses on balancing between two domains. Extending this to multiple domains may introduce additional complexities. In some cases, the data manipulation strategy might force the model to learn from less relevant data, potentially leading to negative transfer between tasks.

Table 1: Millions of network updates for each training procedure using the SAL architecture. Bolded results indicate a decrease in number of network updates for that task to reach 90% success rate. Single env is the results from training a single multi-task policy on Meta-World or Franka Kitchen. MW or FK beside the domain names indicate where that task is from. SAL (MTA) indicates the results of using SAL with the manual task alignment. High FK is the policy update data manipulation experiment with a ratio of 90% Franka Kitchen data and 10% Meta-World data at the start of training. High MW is the opposite.

Task	Single Env	SAL (MTA)	High FK	High MW
Drawer Close (MW)	6.4	10.7	**2.4 (-62.76%)**	11.5
Drawer Open (MW)	42.7	211.2	**24.8 (-42.0%)**	53.9
Reach (MW)	17.1	19.2	**12.7 (-25.88%)**	27.9
Slide Cabinet (FK)	14.9	19.2	**11.5 (-23.0%)**	**4.3 (-71.0%)**
Microwave (FK)	76.8	147.2	**38.3 (-50.13%)**	**71.5 (-6.94%)**

5.3 Future Work

In this work we have highlighted several successful methods for training a single policy across reinforcement learning domains with differing state and action spaces. However, we have also highlighted several limitations of our proposed method, namely the task alignment process and the data manipulation experiments.

In order to align the tasks for training using our SAL policy, we randomly sampled alignments and then trained policies using those alignments. As the number of tasks increases this would be an unusable method. Instead of this method, it may be possible to learn a set of options [17] and compare the trajectories induced by these options across all tasks. If an option induces similar trajectories on the task it was trained on and some new task, they may be similar tasks.

One of the methods that we found increased performance of the SAL policy from both a success rate and sample efficiency perspective was the manipulation of data ratios in the SAL policy update. However, it's possible that this method would only work when the included domains are robotic manipulation tasks. A useful future work could be to explore the usefulness of training on multiple domain tasks types, such as robotic manipulation, navigation, and other continuous control domains similar to [8].

6 Conclusion

In this paper we explored the problem of applying multi-task reinforcement learning algorithms to domains with differing state and action space sizes, in addition to the state and action spaces having differing semantic meaning. We propose the Shared, Domain-Agnostic, Latent (SAL) policy to overcome these issues which leverage separate input translation layers and output action heads for each domain. We explored the usefulness of SAL and the methods that are needed to train an effective policy. We found that by applying these methods, we can increase the sample efficiency of learning in Meta-World and Franka Kitchen domains by up to 70%. We also highlighted some of the limitations and future directions for this work, including a method of determining if two tasks are similar and an investigation into training on multiple tasks with potentially different input types. This work serves as a foundation for further exploration into flexible learning architectures that can bridge the gap between diverse task domains and robot configurations.

Acknowledgments

We are grateful to the anonymous reviewers for their constructive feedback. Resources used in preparing this research were provided, in part, by the Province of Ontario through a Queen Elizabeth II Graduate Scholarship in Science and Technology.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

[2] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, Dec. 2020. ISSN 1476-4687. doi:10.1038/s41586-020-2939-8. URL https://doi.org/10.1038/s41586-020-2939-8.

[3] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, Feb. 2022. ISSN 1476-4687. doi:10.1038/s41586-021-04301-9. URL https://www.nature.com/articles/s41586-021-04301-9. Number: 7897 Publisher: Nature Publishing Group.

[4] T. Chen, A. Murali, and A. Gupta. Hardware Conditioned Policies for Multi-Robot Transfer Learning, Jan. 2019. URL http://arxiv.org/abs/1811.09864. arXiv:1811.09864 [cs].

[5] Q. Zhang, T. Xiao, A. A. Efros, L. Pinto, and X. Wang. Learning Cross-Domain Correspondence for Control with Dynamics Cycle-Consistency, Dec. 2020. URL http://arxiv.org/abs/2012.09811. arXiv:2012.09811 [cs].

[6] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. PARROT: DATA-DRIVEN BEHAVIORAL PRIORS FOR REINFORCEMENT LEARNING. 2021.

[7] D. Tanneberg, K. Ploeger, E. Rueckert, and J. Peters. SKID RAW: Skill Discovery from Raw Trajectories. *IEEE Robotics and Automation Letters*, 6(3):4696–4703, July 2021. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2021.3068891. URL http://arxiv.org/abs/2103.14610. arXiv:2103.14610 [cs].

[8] D. Shah, A. Sridhar, A. Bhorkar, N. Hirose, and S. Levine. GNM: A General Navigation Model to Drive Any Robot, May 2023. URL http://arxiv.org/abs/2210.03370. arXiv:2210.03370 [cs].

[9] N. Bohlinger, G. Czechmanowski, M. Krupka, P. Kicki, K. Walas, J. Peters, and D. Tateo. One Policy to Run Them All: an End-to-end Learning Approach to Multi-Embodiment Locomotion, Sept. 2024. URL http://arxiv.org/abs/2409.06366. arXiv:2409.06366 [cs].

[10] B. Trabucco, M. Phielipp, and G. Berseth. AnyMorph: Learning Transferable Polices By Inferring Agent Morphology, June 2022. URL http://arxiv.org/abs/2206.12279. arXiv:2206.12279 [cs].

[11] T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, and S. Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. *arXiv:1910.10897 [cs, stat]*, June 2021. URL http://arxiv.org/abs/1910.10897. arXiv: 1910.10897.

[12] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters. SHARING KNOWLEDGE IN MULTI-TASK DEEP REINFORCEMENT LEARNING. 2020.

[13] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning, Oct. 2019. URL https://arxiv.org/abs/1910.11956v1.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Aug. 2018. URL http://arxiv.org/abs/1801.01290. arXiv:1801.01290 [cs, stat].

[15] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] G. E. Hinton and S. Roweis. Stochastic Neighbor Embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002. URL https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[17] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi:https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

[18] R. Yang, H. Xu, Y. WU, and X. Wang. Multi-Task Reinforcement Learning with Soft Modularization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4767–4777. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/32cfdce9631d8c7906e8e9d6e68b514b-Paper.pdf.

[19] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[20] M. Cho, W. Jung, and Y. Sung. Multi-task reinforcement learning with task representation method. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022. URL https://openreview.net/forum?id=rV2zaEpNybc.

[21] H. He, C. Bai, K. Xu, Z. Yang, W. Zhang, D. Wang, B. Zhao, and X. Li. Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning. *arXiv:2305.18459*, 2023.

[22] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[23] M. Xu, M. Veloso, and S. Song. ASPiRe: Adaptive Skill Priors for Reinforcement Learning. 2022.

[24] M. Xu, Z. Xu, C. Chi, M. Veloso, and S. Song. XSkill: Cross Embodiment Skill Discovery. 2023.

[25] J. Yang, C. Glossop, A. Bhorkar, D. Shah, Q. Vuong, C. Finn, D. Sadigh, and S. Levine. Pushing the Limits of Cross-Embodiment Learning for Manipulation and Navigation, Feb. 2024. URL `http://arxiv.org/abs/2402.19432`. arXiv:2402.19432 [cs].

[26] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. LEARNING AN EMBEDDING SPACE FOR TRANSFERABLE ROBOT SKILLS. 2018.

[27] W. Huang, I. Mordatch, and D. Pathak. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control, July 2020. URL `http://arxiv.org/abs/2007.04976`. arXiv:2007.04976 [cs, stat].

[28] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei. MetaMorph: Learning Universal Controllers with Transformers, Mar. 2022. URL `http://arxiv.org/abs/2203.11931`. arXiv:2203.11931 [cs].

[29] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Hyq4yhile`.

A Related Work

One of the first comprehensive benchmarks for testing in the multitask RL domain was Meta-World [11]. The benefit of using the Meta-World benchmark is that it has a high degree of shared domain and control structure which allows for efficient learning of distinct but related tasks [11]. Another benefit of Meta-World is the dense reward function available for each individual task. [11] propose several benchmark algorithms for multitask reinforcement learning including Multi-Task Multi-Head Soft Actor Critic (MTMHSAC). The MTMHSAC algorithm modifies the base Soft-Actor Critic algorithm by adding an entropy head for each task, allowing for different levels of exploration per task [11]. In Yu et al. [11] the states are augmented with a one-hot vector that indicates which domain the state belongs to. Other recent approaches include Soft-Modularization which uses the one-hot vector as input to a routing network that outputs probabilities for how data is routed through the policy network [18], Yu et al. [19] developed a method of projecting conflicting gradients onto the same plane thus making network optimization more efficient, Cho et al. [20] developed a variational based method as well as a measure of negative transfer. Recently, He et al. [21] have shown diffusion models to be effective planners and data synthesizer in multitask reinforcement learning settings.

Gupta et al. [13] originally designed Franka Kitchen as a benchmark for algorithms that can solve long-horizon, multitask problems. Franka Kitchen was then modified by [22]. The robotic arm in Franka Kitchen is a 9-DOF Franka robot that is placed in a kitchen domain with many different household kitchen objects. The goal of the domain is to achieve some desired configuration of the objects through manipulation [13].

We formulate our problem with the context of agent-agnostic reinforcement learning. There has been numerous works that take different perspectives on learning policies that can be applied, or quickly adapted, for running on robotics hardware with a different morphology than the policy was trained on. There are two main approaches to these works. The first line of work is interested in leveraging large datasets, with or without actions, and learning how to extract useful skill information from the dataset. [6] uses a dataset of successful trajectories to learn a prior over robotic tasks that can then be applied to unseen new tasks. [7] learns a latent variable model that is able to segment unlabelled trajectories into subtasks used across all tasks in the dataset, with these learned subtasks being transferrable to physical robotics manipulations tasks. Similarly, [23] learns a dictionary of skill priors from expert demonstrations and then trains a policy conditioned on these skill priors enabling effective long-horizon model-free reinforcement learning. XSkill [24] has been proposed for cross embodiment skill discovery, where a dataset of videos across multiple embodiments are used to discover skills, and then the learned skills are transferred to the current embodiment and aligned with the current tasks for downstream skill-conditioned visuomotor policy learning. Lastly [25] trained a single, goal-conditioned, policy across manipulation, navigation, and driving datasets with various embodiments. [25] found that there was an increase in goal completion when combining the manipulation and navigation data, while postulating that the policy must understand where the current state is with respect to it's goal, as well as understanding how to navigate cluttered spaces. We differ from each of these previous works as we aim to do reinforcement learning without any priors, and completely in an online fashion.

In addition to the work that learns on large datasets, there is also a line of work that aims to condition the policy learning on a vector representation that captures task and/or robotic morphology information. [26] was one of the first works to learn a representation of tasks, which allowed for interpolation between learned embeddings for zero-shot transfer of skills. However, when applied to Meta-World, the performance of this method degrades significantly [11]. In order to facilitate cross embodiment learning, [27] proposed to learn a modular policy for each component of an agent's morphology that passes messages from module to module. [28] leverages a transformer architecture to combat the inefficiencies in graph neural networks for incompatible MTRL by injecting explicit morphological information into the model. [10] proposes to infer robot morphology using the reinforcement learning objective directly, instead of a representation learning objective. Recently, [9] proposed a multi-task reinforcement learning method to learn across various quadreped robots.

Concurrently to our work, [9] make use of observation and joint specific information to train a single policy across multiple legged robots. Our work differs from [9] in the types of embodiments, the tasks, and the types of observations. We don't supply the SAL policy with information about specifics of the joint it is controlling. We differ from this line of work, in general, as we don't inject or infer any embodiment information. We aim to share all overlapping task information within the SAL policy network, with the translation layers and action heads handling the cardinalities of the state and action spaces.

B Background information

B.1 Reinforcement Learning

In this work, the Reinforcement Learning (RL) algorithm that we use is Soft-Actor Critic (SAC). SAC is an off-policy RL algorithm that is based on the maximum entropy learning framework [14]. This framework leads the agent to maximize both expected rewards and entropy, which leads the agent to succeed in the task while acting as randomly as possible [14]. In this work there are three sets of parameters to optimize: the soft Q-function $Q_\theta(s_t, a_t)$ with Q parameterized by θ, the policy parameters $\pi_\phi(a_t|s_t)$ where π is parameterized by ϕ, and the entropy penalty coefficient α [14]. The objective for policy optimization is:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D}[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)] \tag{1}$$

with α controlling the entropy penalty coefficient. The coefficient α is learned using the objective:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_\phi}[-\alpha \log \pi_\phi(a_t|s_t) - \alpha \bar{\mathcal{H}}] \tag{2}$$

where $\bar{\mathcal{H}}$ is the minimum target entropy.

Previous work has modified the SAC algorithm to include an entropy term for each of the N tasks to guide exploration in each task individually, as well as to introduce replay buffers per task (Yu et al. [11], Yang et al. [18]).

B.2 Meta-World

The first set of domains used in this work are from Meta-World [11]. Meta-World is a suite of multitask RL and meta-RL domains that consist of a number of robotic manipulation tasks. These domains are subdivided into different sets with any number of domains and different goals. This work focuses on the Multi-Task 10 (MT10) set of domains. In the MT10 set, there are 10 tasks that the RL agent can interact with. Some domains are closely related to each other, such as window open and window close, while other domains are not as closely related to each other, such as pick and place, and window close. This difference in tasks allows for a wide variety of skills to be learned across these domains with robust learning happening due to the number of goals available for each of the individual tasks[11]. Meta-World also provides a dense and smooth reward function to help RL agents learn[11]. The different tasks of the MT10 set of domains can be found in Figure 4.

B.3 Franka Kitchen

The Franka Kitchen domain is the other domain to be used in this work. The Franka Kitchen domain has typically been used in hierarchical RL where the goal is to complete a number of tasks sequentially [13]. This work uses the Franka Kitchen domain in a slightly different manner where each of the available tasks in Franka Kitchen are used individually to create a similar set of tasks to MT10 from Meta-World. This can be verified in Figure 5. Thus, the agent only needs to solve one of the available tasks in a single domain. The default reward function in Franka Kitchen is a sparse reward function where the agent only receives a reward for solving the task [13].

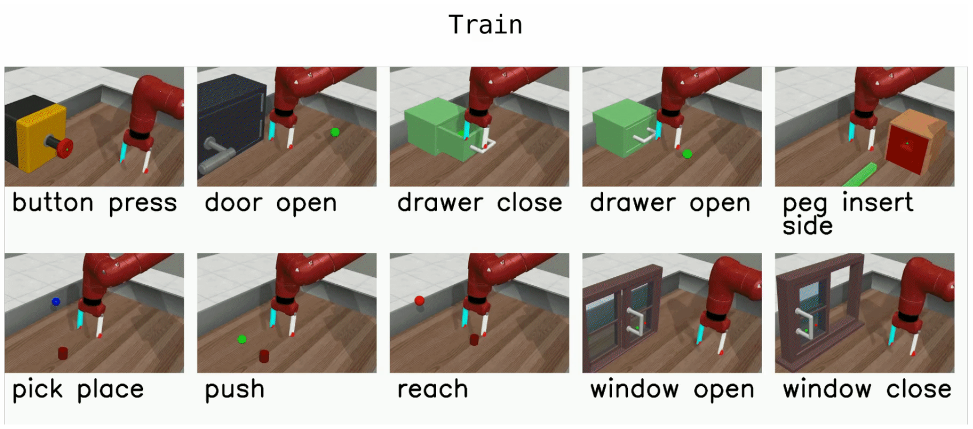


Figure 4: Meta-World MT10 tasks, image from [11].

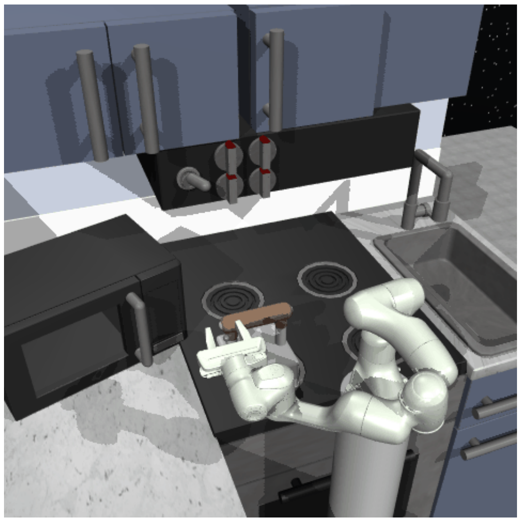


Figure 5: Franka Kitchen domain, image from [29].

C Implementation Details

C.1 Hyperparameters

Codebase is

D Evaluation procedure

We evaluate the performance and transfer capabilities of the RL agent in two domains: Meta World [11] and Franka Kitchen [13](see Section B.2 and B.3). In all experiments, we report the success rate across 100 episodes per task with 50,000 gradient steps each to ensure that our results are statistically significant. These results are reported across 2 different random seeds. For our multitask policy $\pi_\theta(a|s,t)$, the task identifier t is a one-hot encoding of the task to inform the agent what task it is solving.

In addition to the success rate, we also report the number of gradient update steps to our policy that it takes to learn specific tasks. In order to calculate the number of updates it takes to learn a task, we choose a success threshold of 90%, once a policy learns a task past this threshold for the first time we can then calculate the number of network gradient updates it took to learn this task. This value is

Table 2: Dense reward functions used for Franka Kitchen from Meta-World.

Franka Kitchen task	Meta-World Reward Function
Microwave	Door Open
Right Hinge Door	Door Open
Left Hinge Door	Door Open
Light Switch	Sweep
Top Right Burner	Dial Turn
Top Left Burner	Dial Turn
Bottom Right Burner	Dial Turn
Bottom Left Burner	Dial Turn
Slide Cabinet	Push
Kettle	Pick Place

calculated by the following formula: $C * GS * BS$, where C is the current epoch, GS is the number of gradient steps per epoch, and BS is the number of samples of data for this task.

E Reward Function for Franka Kitchen

The Franka Kitchen domain uses a sparse reward function that gives the agent a reward of 0.3 for completing the desired task and 0 otherwise. This limits the ability to do online RL as it is extremely difficult for the agent to learn the correct sequence of actions to complete any of the tasks. To overcome this challenge in creating our baseline approach, we adopt a modified version of the dense reward function that is used in Meta-World for Franka Kitchen. Table 2 shows the mapping between the Franka Kitchen task to be solved and the reward function used from Meta-World to provide dense rewards.

Some slight modifications are made to the reward functions to use them in Franka Kitchen. Both Meta-World and Franka Kitchen were designed using Mujoco. However, the objects that a RL agent interacts with in Meta-World are Mujoco bodies, while most of the Franka Kitchen objects are attached to bodies. The exception to this in Franka Kitchen is the kettle as it is a body itself. To overcome this issue, the reward functions for each task are modified slightly to use different Mujoco sites of the object of interest. For example, the microwave reward function uses the handle site as a method of determining how open or closed the door is. We refer the interested reader to our public implementation for further details on how the reward function was modified to use sites. To denote that a task has been completed, we used a threshold of 5 cm.

Table 3: domains that were aligned across MW and FK are shown. The Button Press Topdown, Window Close, Peg Insert Side, Reach, Drawer Open, Top Right Burner, Top Left Burner, Bottom Right Burner, and Bottom Left Burner tasks all received unique one-hot IDs.

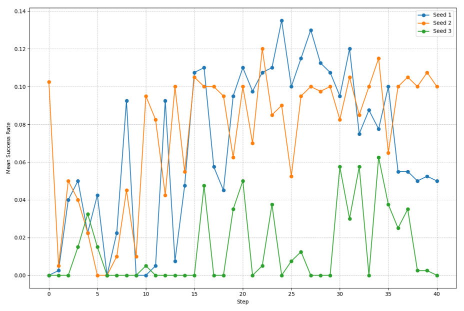
Franka Kitchen task(s)	Meta-World task(s)
Slide Cabinet	Drawer Close
Kettle	Pick Place
Left Hinge Door, Right Hinge Door, Microwave	Door Open
Light Switch	Push

F Implementation and Compute Details

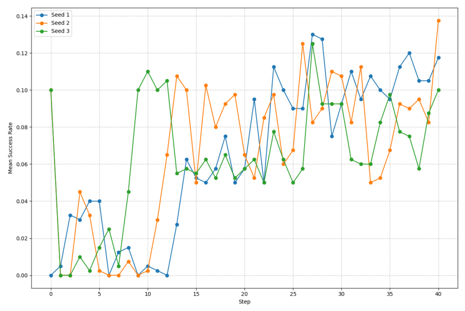
Code will be open-sourced upon acceptance.

G Task Alignment

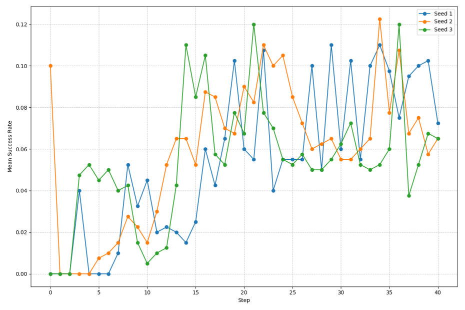
Figure 6 shows the performance of 3 additional seeds of random task alignment, while Figure 7 shows the latent vector alignment after 1 epoch of training, and then after a full experiment run.



(a)



(b)



(c)

Figure 6: Results from 3 additional random seeds of alignment experiments

H Network Updates Data

Table 4: Millions of network updates for each training procedure on Meta-World tasks. This number is based on how many network updates it took to get to a 90% success rate. 0's indicate that the task was not learned in that training configuration.

Task	Single domain	FK&MW(50/50)	High FK	High MW
Door Open	32.0	168.5	46.9	69.1
Drawer Close	6.4	10.7	2.4	11.5
Drawer Open	42.7	211.2	24.8	53.9
Window Open	25.6	185.6	46.9	72.7
Window Close	14.9	155.7	43.7	64.8
Peg Insert Side	0.0	0.0	0.0	0.0
Pick Place	0.0	0.0	0.0	0.0
Push	0.0	0.0	0.0	140.8
Button Press Topdown	49.1	215.5	91.7	110.9
Reach	17.1	19.2	12.7	27.9
Total Network Updates	187.7	966.4	269.1	551.7

Figure 7: T-SNE embeddings from the translation layers (a + b) and the SAL policy (c + d). We find that the policy is able to maintain connections for tasks where there is an increase in sample efficiency.

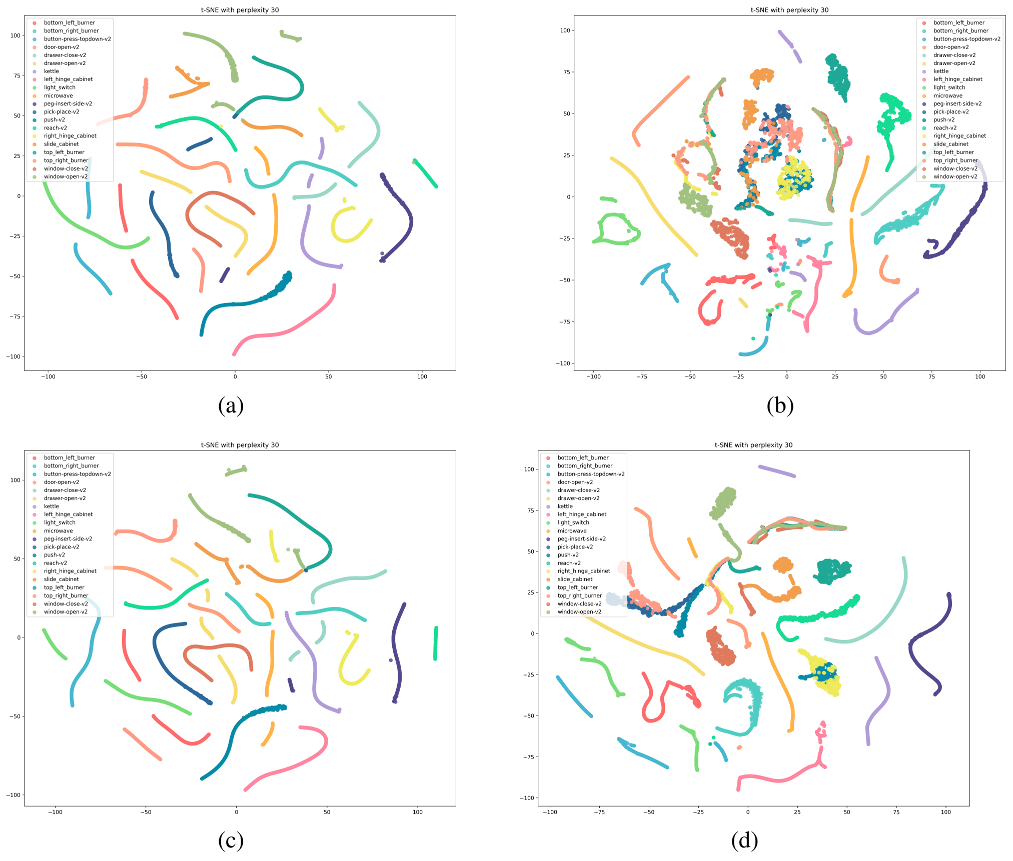


Table 5: Millions of network updates for each training procedure on Franka Kitchen tasks. This number is based on how many network updates it took to get to a 90% success rate. 0's indicate that the task was not learned in that training configuration.

Task	Single domain	FK&MW(50/50)	High FK	High MW
Slide Cabinet	14.9	19.2	11.5	4.3
Microwave	76.8	147.2	38.3	71.5
Top Right Hinge Cabinet	0.0	0.0	0.0	0.0
Top Left Hinge Cabinet	125.9	243.2	0.0	0.0
Top Right Burner	19.2	0.0	0.0	0.0
Top Left Burner	185.6	0.0	0.0	0.0
Bottom Right Burner	6.4	0.0	0.0	68.4
Bottom Left Burner	138.7	0.0	0.0	0.0
Kettle	0.0	0.0	0.0	0.0
Light Switch	14.9	125.9	110.1	41.2
Total Network Updates	582.4	535.5	159.9	185.4