Multi-Source Transfer Learning for Deep Model-Based Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

A crucial challenge in reinforcement learning is to reduce the number of interactions with the environment that an agent requires to master a given task. Transfer learning proposes to address this issue by re-using knowledge from previously learned tasks. However, determining which source task qualifies as optimal for knowledge extraction, as well as the choice regarding which algorithm components to transfer, represent severe obstacles to its application in reinforcement learning. The goal of this paper is to alleviate these issues with modular multi-source transfer learning techniques. Our proposed methodologies automatically learn how to extract useful information from source tasks, regardless of the difference in state-action space and reward function. We support our claims with extensive and challenging cross-domain experiments for visual control.

1 Introduction

Reinforcement learning (RL) agents typically interact with a single unknown environment in order to master a given task. Generally, the number of interactions required to do so is enormous, limiting the applicability of RL to the real world. Recent works on model-based RL (MBRL) address this issue of sample inefficiency by learning a world model, which allows an agent to simulate interactions with complex environments (Ha & Schmidhuber, 2018; Hafner et al., 2019; 2020). However, learning an accurate model of the environment from scratch still requires a substantial amount of interactions. A common approach for enhancing sample efficiency in the supervised learning domain is transfer learning (Yosinski et al., 2014; Sharif Razavian et al., 2014; Zhuang et al., 2020). By re-using the parameters of a neural network that were fit to some previous task (referred to as source task) for learning some other task (referred to as target task), one can reduce the number of samples required substantially compared to fitting randomly initialized parameters. However, the performance of transfer learning highly depends on the choice of the source task, which is often based on the intuition of the designer (Taylor & Stone, 2009b). As RL environments can differ in several fundamental aspects, the application of transfer learning in this domain is especially challenging and has received little attention. In this paper, we address this issue and propose to alleviate it with multi-source transfer learning techniques. Rather than re-using information from a single source task, multi-source transfer learning proposes to extract knowledge from multiple source tasks. Our proposed methodologies autonomously learn to extract useful information from a collection of source tasks, regardless of the differences between environments, alleviating the necessity of selecting an optimal source task. We apply our proposed techniques to the state-of-the-art world model-based algorithm for visual continuous control tasks, Dreamer (Hafner et al., 2019). We focus on world model-based algorithms, as learning reward, dynamics, policy, and value models in a compact latent space of a world model has shown promising results for the application of transfer learning, yet little research has been conducted in this direction (Zhu et al., 2020). A major obstacle of applying transfer learning to these types of algorithms is that they are composed of several different components. Each component may or may not benefit from different types of transfer learning, dependent on their objective function and the differences between

source and target tasks. We provide valuable insights and empirical evidence on the transferability of such components, by applying multi-source transfer learning to Dreamer in a modular fashion, in addition to introducing a novel type of transfer learning that allows the fractions of parameters to be transferred. We extensively evaluate the proposed methods in challenging cross-domain transfer learning experiments. We consider two multi-source transfer learning settings: a single agent that masters multiple tasks, and multiple individual agents that each master a single task. We provide techniques that allow cross-domain transfer learning for both of these settings, which are also applicable in single-source transfer learning settings. However, we encourage the usage of multi-source transfer learning, in order to avoid manual selection of an optimal source task. The contributions are summarized as follows:

- **Fractional transfer learning**: We introduce a novel type of transfer learning that allows fractions of parameters to be transferred, as an alternative to discarding information by random initialization, resulting in substantial performance improvements.
- **Modular multi-task transfer learning**: We show that training a single world model-based agent on multiple tasks simultaneously and transferring its components in a modular fashion with different types of transfer learning, results in enhanced sample efficiency and performance compared to learning from scratch.
- **Meta-model transfer learning**: We propose a multi-source transfer learning approach that allows models from several individual agents trained in different environments to be combined and transferred in a shared feature space. These components produce additional input signals for the corresponding model of the target agent, allowing us to construct a meta-model that learns to weigh the usefulness of these signals for learning the target task, resulting in significant performance gains when applied to merely one model.

2 Preliminaries

We formalize an RL problem as a Markov Decision Process (MDP) (Bellman, 1957), which is a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, P the transition function, and R the reward function. For taking a given action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, $P(s'|s, a)$ denotes the probability of transitioning into state $s' \in \mathcal{S}$, and $R(r|s, a)$ yields an immediate reward r. The state and action spaces define the domain of the MDP. The objective of RL is to find an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected cumulative reward. The expected cumulative reward is defined as $G_t = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$, where $\gamma \in [0, 1)$ represents the discount factor, and t the time-step.

The general concept of transfer learning aims to enhance the learning of some *target* task by re-using information obtained from learning some *source* task. Multi-source transfer learning aims to enhance the learning of a target task by re-using information from a *set* of source tasks. In RL, a task is formalized as an MDP[1] M with some optimal policy π^*. As such, in multi-source transfer learning for RL we have a collection of N source MDPs $\mathcal{M} = \{(\mathcal{S}_i, \mathcal{A}_i, P_i, R_i)\}_{i=1}^{N}$, each with some optimal policy π_i^*. Let $M = (\mathcal{S}, \mathcal{A}, P, R)$ denote some target MDP with optimal policy π^*, where M is different from all $M_i \in \mathcal{M}$ with regards to $\mathcal{S}, \mathcal{A}, P$,or R Multi-source transfer learning for reinforcement learning aims to enhance the learning of π^* by reusing information obtained from learning $\pi_1^*, .., \pi_N^*$. When $N = 1$ we have single-source transfer learning.

In this paper, we use Dreamer as a reference for deep MBRL algorithms (Hafner et al., 2019). MBRL algorithms learn to model P and R internally. Dreamer learns to model these functions in a compact latent space learned from visual observations, referred to as a world model Ha & Schmidhuber (2018). As such, interactions with environments with high-dimensional observations can be imagined in a computationally efficient manner, which is used to facilitate sample-efficient policy learning. The policy is used to collect

[1]We use the terms MDP, task, and environment interchangeably in this paper.

data from the environment, which is used for learning the world model. The world model consists of four components: a representation model $p_\theta(s_t|s_{t-1}, a_{t-1}, o_t)$, observation model $q_\theta(o_t|s_t)$, reward model $q_\theta(r_t|s_t)$, and transition model $q_\theta(s_t|s_{t-1}, a_{t-1})$, where p denotes distributions that generate real environment samples, q denotes distributions approximating those distributions in latent space, and θ denotes the jointly optimized parameters of the models. At a given timestep t, the representation model maps a visual observation o_t together with the previous latent state s_{t-1} and previous action a_{t-1} to latent state s_t. The transition model learns to predict s_t from s_{t-1} and a_{t-1}. The reward model learns to predict the reward r_t corresponding to s_t. The observation model reconstructs s_t to match o_t, providing the learning signal for learning the feature space. This world model is called the recurrent state space model (RSSM), and we refer the reader to Hafner et al. (2018) for further details. In order to imagine a trajectory $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ of length H, where τ denotes the imagined time index, the representation model maps an initial observation o_t to latent state s_τ, which is combined with action a_τ yielded by the policy, to predict $s_{\tau+1}$ using the transition model. The reward model then predicts the corresponding reward $r_{\tau+1}$, which is used for policy learning.

In order to learn a policy, Dreamer makes use of an actor-critic approach, where the action model $q_\phi(a_\tau|s_\tau)$ implements the policy, and the value model $v_\psi(s_t) \approx \mathbb{E}_{q(\cdot|s_\tau)}(\sum_{\tau=t}^{t+H} \gamma^{\tau-t} r_\tau)$ estimates the expected reward that the action model achieves from a given state s_τ. Here ϕ and ψ are neural network parameters for the action and value model respectively, and γ is the discount factor. The reward, value, and actor models are implemented as Gaussian distributions parameterized by feed-forward neural networks. The transition model is a Gaussian distribution parameterized by a Gated Recurrent United (GRU; Bahdanau et al. (2014)) followed by feed-forward layers. The representation model is a variational encoder (Kingma & Welling, 2013; Rezende et al., 2014) combined with the GRU, followed by feed-forward layers. The observation model is a transposed convolutional neural network (CNN; LeCun et al. (2015)).

3 Related Work

In supervised learning, parameters of a neural network are transferred by either freezing or retraining the parameters of the feature extraction layers, and by randomly initializing the parameters of the output layer to allow adaptation to the new task (Yosinski et al., 2014; Sabatelli et al., 2018; Cao et al., 2021). Similarly, we can transfer policy, value, and reward models, depending on the differences of the state-action spaces, and reward functions between environments (Carroll & Peterson, 2002; Schaal et al., 2004; Fernández & Veloso, 2006; Rusu et al., 2016; Zhang et al., 2018). We can also transfer autoencoders, trained to map observations to latent states (Chen et al., 2021). Experience samples collected during the source task training process can also be transferred to enhance the learning of the target task (Lazaric et al., 2008; Tirinzoni et al., 2018). The transferability of deep model-free RL algorithms doesn't appear to be promising, as was recently shown by Sabatelli & Geurts (2021). However, they are suitable for distillation techniques, where a new neural network learns to predict the mappings of inputs to outputs of a pre-trained network (Hinton et al., 2015; Parisotto et al., 2015; Rusu et al., 2015). By sharing a distilled policy across multiple agents learning individual tasks, Teh et al. (2017) obtain robust sample efficiency gains. In order to address the major obstacle of optimal task selection, Ammar et al. (2014) introduced an autonomous similarity measure for MDPs based on restricted Boltzmann machines, though it assumes the MDPs are within the same domain. García-Ramírez et al. (2021) propose to select the best source models among multiple model-free models using a regressor.

This paper focuses on transfer learning for deep MBRL, where we also need to consider the dynamics model. Recent works showed that the dynamics model can be fully transferred between different domains if the environments are sufficiently similar (Eysenbach et al., 2020; Rafailov et al., 2021). Landolfi et al. (2019) perform a multi-task experiment, where the dynamics model of an MBRL agent is transferred to several novel tasks sequentially, and show that this results in significant gains of sample efficiency. PlaNet (Hafner et al., 2018) was used in a multi-task experiment, where the same agent was trained on tasks of

six different domains simultaneously using a single world model (Ha & Schmidhuber, 2018). Similarly, In Plan2Explore (Sekar et al., 2020) the authors show that a single global world model can be trained task-agnostically, after which a Dreamer (Hafner et al., 2019) agent can use the model to zero-shot or few-shot learn a new task within the same domain. Dreamer is not to be confused with DreamerV2 (Hafner et al., 2020), which is essentially the same algorithm adapted for discrete domains. Unlike previous works, we investigate multi-source transfer learning with a focus on world model-based algorithms, by introducing multiple techniques that enhance sample efficiency without the need of selecting an optimal source task.

4 Methods

Here we present the main contributions of this work. First, we propose multi-task learning as a multi-source transfer learning origin, combined with modular and fractional transfer learning. We follow this with an alternative setting, where we propose to transfer components of multiple individual agents utilizing a shared feature space and meta-model instead.

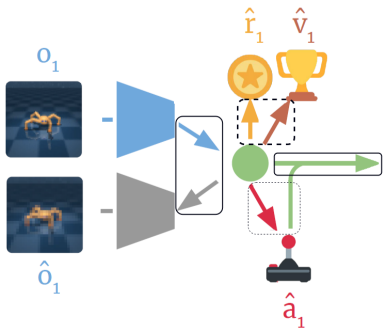

Figure 1: We apply transfer learning to Dreamer in a modular fashion. Arrows represent the parameters of a model. The representation, observation and transition model are fully transferred (——). The reward and value models are fractionally transferred (- - -). The action model and the action-input parameters of the transition model are randomly initialized (- - -).

4.1 Multi-Task Transfer Learning

In this section, we describe how we train a single agent on multiple MDPs simultaneously, and introduce a novel type of transfer learning that allows fractions of parameters to be transferred. We then discuss what type of transfer learning is most suitable for each component of the agent.

4.1.1 Simultaneous Multi-Task Learning

We propose to train single agents facing multiple unknown environments simultaneously, each with different state-action spaces and reward functions, in order to use them as multi-source transfer learning origins. We train the multi-task RL agents in a similar fashion to the experiments done by Hafner et al. (2018), where the action dimension of each task is padded with unused elements to the size of the task with the largest action dimension. We are required to have uniform action dimensions across all source environments, as we use a single policy model. The agent collects one episode of each task in collection phases in order to acquire a balanced amount of experiences in the replay buffer.

4.1.2 Fractional Transfer Learning

Typically, one can either fully transfer the parameters of a neural network layer, or randomly initialize the parameters. Feature extraction layers of neural networks are generally fully transferred (Sermanet et al., 2013). The output layer of a neural network is often randomly initialized, in order to prevent overfitting. However, this means that we discard all information contained in the parameters of the output layer. We propose a simple alternative: *fractional transfer learning* (FTL), which allows us to transfer fractions of parameters. Rather than discarding all information, this technique allows us to transfer a portion of previously obtained information, without risking overfitting. Let θ_T denote target parameters, θ_ϵ randomly initialized weights, α the fraction parameter, and θ_S source parameters, then we can do FTL by $\theta_T = \theta_\epsilon + \alpha\theta_S$. That is, we add a fraction of the source parameters to the randomly initialized weights.

4.1.3 Modular Transfer learning

Modern deep MBRL algorithms often consist of several components, each relating to different elements of an MDP. Therefore, in order to transfer the parameters of such architectures, we need to consider transfer learning on a modular level. Using Dreamer as a reference, we discuss what type of transfer learning each component benefits most from. We consider three types of transfer learning: random initialization, FTL, and full transfer learning. We found in initial experiments that fully transferring all feature extraction layers of each component results in the best learning enhancements, and found that the output layer mainly impacts the transfer learning performance. Therefore, we only discuss transferring parameters of output layers of the multi-task agents.

First, we consider the action model, which implements the policy. As action elements don't match across different environments, our initial experiments showed, as anticipated, that both full transfer and FTL across environments with different action spaces results in detrimental performance drops compared to random initialization.

Next, we found that fractionally transferring parameters of the reward and value models can result in substantial performance gains (Appendix E). In this paper, we apply our methods to MDPs that have similar reward functions, meaning the parameters of these models consist of transferable information that enhances the learning of a target task. This demonstrates the major benefits of FTL, as the experiments also showed that fully transferring these parameters has a detrimental effect on learning (Appendix D).

Finally, initial experiments showed that fully transferring the parameters of the representation, observation, and transition model results in the best performance improvements for most environments. As we are dealing with visually similar environments, the generality of convolutional features allows full transfer of the representation and observation parameters (Chen et al., 2021). Additionally, when environments share similar physical laws (e.g. a physics engine) transition models can often be fully transferred, provided that we reset the weights connected to actions (Taylor & Stone, 2009a; Landolfi et al., 2019).

4.2 Multiple-Agent Transfer Learning

In this section, we consider an alternative multi-source transfer learning setting, where we have access to the parameters of multiple individual agents that have learned the optimal policy for different tasks. Transferring components from agents trained in different environments represents a major obstacle for multi-source transfer learning. To the best of our knowledge, we are the first to propose a solution that allows the combination of models from agents trained in different environments, from which the most relevant information can autonomously be extracted for a given target task.

4.2.1 Universal Feature Space

As the focus of this paper is on world model-based agents, we are required to facilitate a shared feature space across the agents when combining and transferring their components. Hence, we introduce a *universal feature space* (UFS) inspired by SEER (Chen et al., 2021), where the authors show that encoders of converged autoencoders can be frozen and reused in visually similar environments due to the generality of convolutional features. In this paper we make use of two different types of environments: locomotion and pendula tasks (Figure 3). Therefore, we decide to train a single agent simultaneously in one locomotion and one pendulum environment (as described in Section 4.1.1), such that we learn convolutional features for both types of environments. After training, we freeze and reuse this agent's encoder across both source and target agents in order to train them within a UFS. Note that we do not transfer and freeze the other RSSM components, as this would prevent the Dreamer agent from learning new reward and transition functions. As Dreamer learns via reconstruction, the encoder is the main component of the representation model responsible for constructing the feature space.

4.2.2 Meta-Model Transfer Learning

When an agent makes use of the UFS for training in a given target task, we can transfer components from agents that each mastered tasks using the UFS in different environments. We propose to accomplish this by introducing *meta-model transfer learning* (MMTL). For a given component of the target agent, we assemble the same component from all source agents into an ensemble of frozen components, which we connect to the component of the target agent. In addition to the usual input element(s), this also provides information signals from each of the frozen source components as a means of transfer learning.

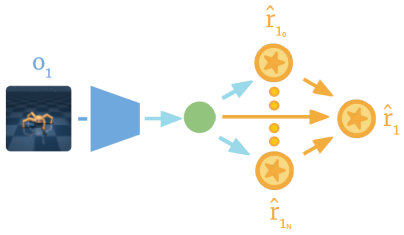

Figure 2: Illustration of meta-model transfer learning applied to Dreamer's reward model.

Let $m_\Theta(y|x)$ denote a neural network component with parameters Θ, some input x, and some output y, belonging to the agent that will be trained on target MDP M. Let $m_{\theta_i}(y|x)$ denote the same component belonging to some other agent i with *frozen* parameters θ_i that were fit to some source MDP M_i, where $i \in N$, and N denotes the number source MDPs on which a separate agent was trained on from the set of source MDPs $\mathcal{M}$. In MMTL, we modify $m_\Theta(y|x)$, such that we get:

$$m_\Theta(\, y \,|\, x,\ m_{\theta_0}(y|x),\ ...,\ m_{\theta_N}(y|x)\,)$$

where all models were trained within the same UFS. Intuitively, information signals provided by the source models can be used to enhance the learning process of m_Θ. For instance, assume the objective of M_i is similar to the objective in M. In that case, if we use MMTL for the reward model, m_Θ can autonomously learn to utilize the predictions of m_{θ_i} via gradient descent. Similarly, it can learn to ignore the predictions of source models that provide irrelevant predictions. As we are dealing with locomotion and pendula environments in this paper, we choose to apply MMTL to the reward model of Dreamer (Figure 2), as the locomotion MDPs share a similar objective among each other whilst the pendula environments will provide irrelevant signals for the locomotion environments and vice versa. As such, we can evaluate whether our approach can autonomously learn to utilize and ignore relevant and irrelevant information signals respectively. Note that in the case of Dreamer's reward model, y corresponds to a scalar Gaussian parameterized by a mean μ and unit variance, from which the mode is sampled. Hence, in our case, y corresponds to μ, and x corresponds to some latent state s. As such, the target agent will make use of the same frozen encoder used by the source agents, in addition to using a reward meta-model. We don't transfer any other components of the architecture in order to be able to observe the isolated effect of the reward meta-model.

5 Experiments

We evaluate the proposed methods using Dreamer[2] on 6 continuous control tasks: Hopper, Ant, Walker2D, HalfCheetah, InvertedPendulumSwingup, and InvertedDoublePendulumSwingup (Figure 3). A detailed description of the differences between the MDPs can be found in Appendix A. We perform experiments for both multi-task (referred to as FTL) and multiple-agent (referred to as MMTL) transfer learning settings.

For each method, we run multi-source transfer learning experiments using a different set of N source tasks for each of the target environments (see Appendix B for the combinations used). The selection of source tasks for a given set was done such that each source set for a given target environment includes at least one task from a different environment type, i.e. a pendulum task for a locomotion task and vice versa. Similarly, each source set contains at least one task of the same environment type. We also ran

[2]This work builds upon the code base of Dreamer: `https://github.com/danijar/dreamer`.

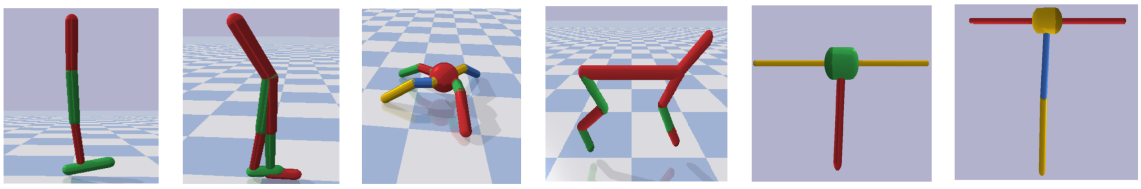

Figure 3: Visualization of the tasks learned in the experiments: Hopper, Walker2D, Ant, HalfCheetah, InvertedPendulumSwingup, and InvertedDoublePendulumSwingup. Each of the MDPs differ in all elements $(\mathcal{S}, \mathcal{A}, R, P)$, and are used in multi-source transfer learning experiments as both source and target tasks.

preliminary experiments (3 random seeds) for sets consisting of $N = [2, 3]$ in order to observe potential performance differences that result from different N, but we found no significant differences (see Appendix C). To demonstrate that our methods can autonomously extract useful knowledge from a set of source tasks that includes at least one irrelevant source task, we apply our methods to the most challenging setting ($N = 4$) for 9 random seeds. We used a single Nvidia V100 GPU for each training run, taking about 6 hours per 1 million environment steps.

For each run, we train the FTL and MMTL target agents for 1 million environment steps and compare them to a baseline Dreamer agent that learns from scratch for 1 million environment steps. FTL is evaluated by training multi-task agents for 2 million environment steps for a single run, after which we transfer the parameters to the target agent as described in Section 4.1.3. We use a fraction of $\alpha = 0.2$ for FTL, as we observed in preliminary experiments the largest performance gains occur in a range of $\alpha \in [0.1, 0.3]$ (see Appendix E). For creating the UFS for MMTL, we train a multi-task agent on the Hopper and InvertedPendulum task for 2 million environment steps. We evaluate MMTL by training a single agent on each task of the set of source environments for 1 million environment steps (all using the same UFS), after which we transfer their reward models to the target agent as described in Section 4.2.2.

The overall aggregated return of both FTL and MMTL is reported in Table 1, which allows us to take jumpstarts into account in the results. The aggregated return of the final 10% (1e5) environment steps can be found in Table 2, allowing us to observe final performance improvements. Figure 4 shows the corresponding learning curves.

6 Discussion

We now discuss the results of each of the proposed methods individually and reflect on the overall multi-source transfer learning contributions of this paper. We would like to emphasize that we don't compare FTL and MMTL to each other, but to the baseline, as they are used for two entirely different settings. Additionally, although the proposed techniques are applicable to single-source transfer learning settings as well, empirical performance comparisons between multi-source and single-source transfer learning are outside of the scope of this paper, and we leave this for future work.

Multi-source transfer learning The transfer learning results show that the proposed multi-source transfer learning methods for both multi-task (FTL) and multiple-agent (MMTL) settings result in positive transfers for 5 out of 6 environments. We observe jumpstarts, overall performance improvements, and/or final performance gains. This shows that our proposed methods allow agents to autonomously extract useful information stemming from source agents trained in MDPs with different state-action spaces, reward functions, and transition functions. We would like to emphasize that for each of the environments, there were at most two environments that could provide useful transfer knowledge. Nevertheless, our methods are

Table 1: Overall average episode return of 1 million environment steps for FTL and MMTL, where parameters of 4 source tasks were transferred. We compare to a baseline Dreamer agent that learns from scratch. Bold results indicate the best performance across the methods and baseline for a given task.

Task	FTL	MMTL	Baseline
HalfCheetah	1490 ± 441	**1882 ± 390**	1199 ± 558
Hopper	3430 ± 2654	**4025 ± 3404**	2076 ± 2417
Walker2D	**1637 ± 2047**	847 ± 1533	676 ± 1101
InvPend	**761 ± 68**	705 ± 79	667 ± 121
InvDbPend	**1235 ± 130**	1198 ± 100	1184 ± 89
Ant	681 ± 591	1147 ± 922	**1148 ± 408**

Table 2: Average episode return of the final 1e-5 environment steps for FTL and MMTL, where parameters of 4 source tasks were transferred. We compare to a baseline Dreamer agent that learns from scratch. Bold results indicate the best performance across the methods and baseline for a given task.

Task	FTL	MMTL	Baseline
HalfCheetah	2234 ± 302	**2458 ± 320**	1733 ± 606
Hopper	5517 ± 4392	**7438 ± 4157**	3275 ± 3499
Walker2D	**2750 ± 2702**	1686 ± 2329	1669 ± 1862
InvPend	**879 ± 17**	872 ± 20	875 ± 23
InvDbPend	**1482 ± 162**	1370 ± 106	1392 ± 115
Ant	1453 ± 591	1811 ± 854	**1901 ± 480**

still able to identify the useful information and result in a positive transfer with 4 source tasks. For the Ant testing environment, we observe negative transfers when directly transferring parameters in the multi-task setting, which logically follows from the environment being too different from all other environments in terms of dynamics, as we fully transfer the transition model. We observe that MMTL does not suffer significantly from this difference in environments, as we don't transfer the transition model in that case.

Fractional transfer learning Transferring parameters in a modular fashion of world model-based agents that were simultaneously trained on multiple tasks overall results in performance improvements. For the single pendulum environment, we observe significant jumpstarts (see Figure 4), despite the set of four source tasks merely containing one environment with useful transfer information. Transferring fractions of parameters plays a major role in these performance improvements as observed in preliminary experiments (Appendix E). However, one drawback of FTL is that it introduces a tunable fraction parameter α, where the optimal fraction can differ per environment and composition of source tasks. We observed the most performance gains in a range of $\alpha \in [0.1, 0.3]$, and for larger fractions the overall transfer learning performance of the agent would degrade. In terms of deep RL hyperparameters, this can be considered a reasonably short tuning range.

Meta-model transfer learning We observe substantial performance gains for MMTL by merely providing additional information signals to the reward model, next to transferring weights of the encoder model. To the best of our knowledge, we are the first to successfully combine individual components of multiple individual agents trained in different environments as a multi-source transfer learning technique that results in positive transfers, which is autonomously accomplished through gradient descent. In addition, the usage of a UFS (which includes a frozen encoder) saves computation.

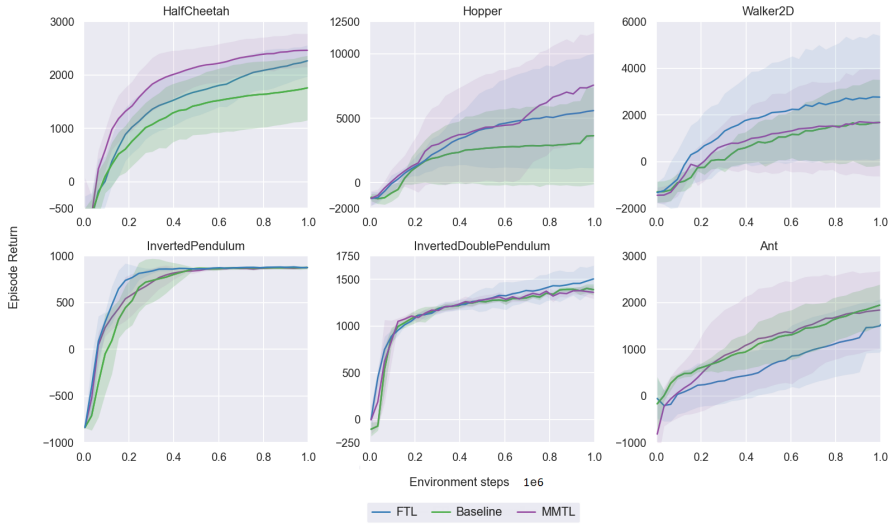

Figure 4: Learning curves for average episode return of 1 million environment steps for FTL and MMTL using 4 source tasks. Shaded areas represent standard deviation across the 9 runs.

7 Conclusion

In this paper, we introduced several techniques that enable the application of multi-source transfer learning to deep MBRL algorithms. Two major obstacles in applying transfer learning to the RL domain are selecting an optimal source task and the choice regarding which type of transfer learning to apply to individual components. We addressed these issues by proposing two modular multi-source transfer learning techniques, each applicable to a different cross-domain multi-source situation: a single agent that mastered several tasks, and multiple agents that mastered a single task.

First, we introduced a novel type of transfer learning, which allows fractions of parameters to be transferred, as opposed to discarding information as commonly done by randomly initializing a layer of a neural network. We used this type of transfer learning as an option in an insightful discussion concerning what type of transfer learning deep MBRL algorithm components benefit from. The conclusions of this discussion were empirically validated in the multi-task transfer learning setting, which both showed that the modular transfer learning decisions result in significant performance improvements, as well as that we can autonomously extract useful information from multiple different source tasks.

Next, we showed that by learning a universal feature space, we enable the combination and transfer of individual components from agents trained in environments with different state-action spaces, and reward functions. We extend this concept by introducing meta-model transfer learning, which leverages the predictions of models trained by different agents in addition to the usual input signals, as a multi-source transfer learning technique for multiple-agent settings. This results in significant sample efficiency gains in challenging cross-domain experiments.

Our aim is to bring RL closer to real-world application and we believe that the techniques introduced in this paper represent an important step toward this direction, by circumventing crucial limitations of transfer learning in RL. Given the flexibility and modularity of the proposed techniques, we believe they are sufficiently general to potentially be applicable to components of other types of deep RL algorithms.

8 Reproducibility

We ensure reproducibility by explaining the proposed methodologies as clearly and in as much detail as possible. First, we described the components of Dreamer that we are applying transfer learning techniques to and how they are implemented in Section 2. For the multi-task setting, we describe how to train an agent on multiple domains in Section 4.1.1, how to make use of fractional transfer learning in Section 4.1.2, and how we apply transfer learning to each of Dreamer's components in Section 4.1.3. Similarly, we describe in detail how we create a universal feature space in Section 4.2.1 and provide a detailed description of our proposed meta-model transfer learning approach and how it is implemented in the Dreamer framework in Section 4.2.2. Moreover, for reproducibility of results, we describe all settings used in the experiments and what hardware was used in Section 5. Finally, we provide anonymized code examples of how to apply the proposed techniques to Dreamer at `https://anonymous.4open.science/r/transfer-dreamer-7308/`README.md.

References

Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL `https://arxiv.org/abs/1409.0473`.

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

Zhangjie Cao, Minae Kwon, and Dorsa Sadigh. Transfer reinforcement learning across homotopy classes. *CoRR*, abs/2102.05207, 2021. URL `https://arxiv.org/abs/2102.05207`.

J L Carroll and Todd S Peterson. Fixed vs. dynamic Sub-Transfer in reinforcement learning. *ICMLA*, 2002.

Lili Chen, Kimin Lee, Aravind Srinivas, and Pieter Abbeel. Improving computational efficiency in visual reinforcement learning via stored embeddings. *arXiv preprint arXiv:2103.02886*, 2021.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. *arXiv preprint arXiv:2006.13916*, 2020.

F Fernández and M M Veloso. Reusing and building a policy library. *ICAPS*, 2006.

Jesús García-Ramírez, Eduardo Morales, and Hugo Jair Escalante. Multi-source transfer learning for deep reinforcement learning. *Lecture Notes in Computer Science Pattern Recognition*, pp. 131–140, 2021. doi: 10.1007/978-3-030-77004-4_13.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018. URL `http://arxiv.org/abs/1811.04551`.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Nicholas C Landolfi, Garrett Thomas, and Tengyu Ma. A model-based approach for sample-efficient multi-task reinforcement learning. *arXiv preprint arXiv:1907.04964*, 2019.

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 544–551, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390225.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. Visual adversarial imitation learning using variational models, 2021.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1278–II–1286. JMLR.org, 2014.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

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.

Matthia Sabatelli and Pierre Geurts. On the transferability of deep-q networks. In *Deep Reinforcement Learning Workshop of the 35th Conference on Neural Information Processing Systems*, 2021.

Matthia Sabatelli, Mike Kestemont, Walter Daelemans, and Pierre Geurts. Deep transfer learning for art classification problems. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.

Stefan Schaal, Auke Jan Ijspeert, Aude Billard, Sethu Vijayakumar, and Jean-Arcady Meyer. Estimating future reward in reinforcement learning animats using associative learning. In *From animals to animats 8: Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior*, pp. 297–304. MIT Press, 2004.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR, 2020.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009a.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009b.

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.

Andrea Tirinzoni, Andrea Sessa, Matteo Pirotta, and Marcello Restelli. Importance weighted transfer of samples in reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4936–4945. PMLR, 10–15 Jul 2018.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

Amy Zhang, Harsh Satija, and Joelle Pineau. Decoupling dynamics and reward for transfer learning. *arXiv preprint arXiv:1804.10689*, 2018.

Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

A Environment Descriptions

In this paper locomotion and pendulum balancing environments from PyBullet (Coumans & Bai, 2016–2021) are used for experiments. The locomotion environments have as goal to walk to a target point that is distanced 1 kilometer away from the starting position as quickly as possible. Each environment has a different entity with different numbers of limbs and therefore has different state-action spaces, and transition functions. The reward function is similar, slightly adapted for each entity as the agent is penalized for certain limbs colliding with the ground. The pendula environments have as objective to balance the initially inverted pendulum upwards. The difference between the two environments used is that one environment has two pendula attached to each other. This environment is not included in the PyBullet framework for swing-up balancing, which we, therefore, implemented ourselves. The reward signal for the InvertedPendulumSwingup for a given observation o is:

$$r_o = \cos\Theta \tag{1}$$

where Θ is the current position of the joint. For the InvertedDoublePendulumSwingup a swing-up task, we simply add the cosine of the position of the second joint Γ to Equation 1:

$$r_f = \cos\Theta + \cos\Gamma \tag{2}$$

As such, these two environments also differ in state-action spaces, transition functions, and reward functions.

B Transfer Learning Task Combinations

In Table 3 the combinations of source tasks and target tasks can be viewed that were used for the experiments in this paper. That is, they correspond to the results of Appendix C and Section 5.

Table 3: Source-target combinations used for FTL and MMTL experiments, using environments HalfCheetah (Cheetah), Hopper, Walker2D, InvertedPendulumSwingup (InvPend), InvertedDoublePendulumSwingup (InvDbPend), and Ant.

Target	2 Tasks	3 Tasks	4 Tasks
Cheetah	Hopper, Ant	+Walker2D	+InvPend
Hopper	Cheetah, Walker2D	+Ant	+InvPend
Walker2D	Cheetah, Hopper	+Ant	+InvPend
InvPend	Cheetah, InvDbPend	+Hopper	+Ant
InvDbPend	Hopper, InvPend	+Walker2D	+Ant
Ant	Cheetah, Walker2D	+ Hopper	+InvPend

C Preliminary Experiments for 2 and 3 Source Tasks

In Table 4 and Table 5 the results for training on, and transferring, 2 and 3 source tasks as described in Section 5 can be found for both FTL and MMTL, showing the overall average episode return and the episode return for the final 1e-5 environment steps respectively. In Figure 5 the corresponding learning curves can be found. These experiments are the averages and standard deviation across 3 seeds for FTL, MMTL, and a baseline Dreamer trained from scratch. As just 3 seeds were used for these experiments, they are not conclusive. However, they do show that regardless of the number of source tasks, our methods can extract useful information and enhance the performance compared to the baseline.

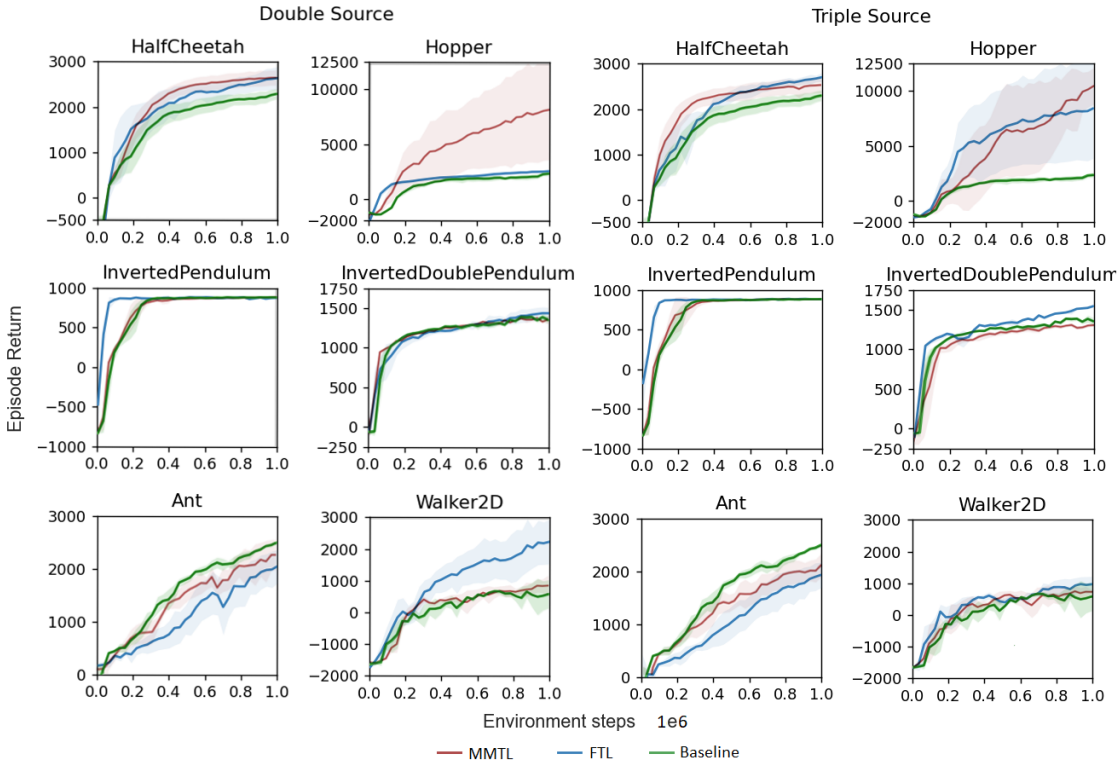

Figure 5: Preliminary experiments: average episode return for 3 random seeds obtained across 1 million environment steps by MMTL (red), FTL (blue), and a vanilla Dreamer (green), where for FTL $\alpha = 0.2$ is used. FTL and MMTL both receive transfer from a combination of 2 (double) and 3 (triple) source tasks. Shaded areas represent standard deviation across the 3 runs.

Table 4: Preliminary experiments: overall average episode return with of FTL using $\alpha = 0.2$ and MMTL for the reward model. Parameters of 2 and 3 source tasks are transferred to the HalfCheetah, Hopper, Walker2D, InvertedPendulumSwingup (InvPend), InvertedDoublePendulumSwingup (InvDbPend), and Ant tasks, and compared to a baseline Dreamer agent that learns from scratch. Bold results indicate the best performance across the methods and baseline for a given task.

	Fractional Transfer Learning		**Meta-Model Transfer Learning**		
Task	2 task	3 task	2 task	3 task	**Baseline**
HalfCheetah	1982 ± 838	1967 ± 862	2057 ± 851	**2078 ± 721**	1681 ± 726
Hopper	1911 ± 712	**5538 ± 4720**	5085 ± 4277	5019 ± 4813	1340 ± 1112
Walker2D	**1009 ± 1254**	393 ± 813	196 ± 860	233 ± 823	116 ± 885
InvPend	874 ± 121	**884 ± 20**	731 ± 332	740 ± 333	723 ± 364
InvDbPend	1209 ± 280	**1299 ± 254**	1214 ± 230	1120 ± 327	1194 ± 306
Ant	1124 ± 722	1052 ± 687	1423 ± 788	1366 ± 687	**1589 ± 771**

Table 5: Preliminary experiments: average episode return of the final 1e5 environment steps of FTL using $\alpha = 0.2$ and MMTL for the reward model. Parameters of 2 and 3 source tasks are transferred to the HalfCheetah, Hopper, Walker2D, InvertedPendulumSwingup (InvPend), InvertedDoublePendulumSwingup (InvDbPend), and Ant tasks, and compared to a baseline Dreamer agent that learns from scratch. Bold results indicate the best performance across the methods and baseline for a given task.

	Fractional Transfer Learning		**Meta-Model Transfer Learning**		
Task	2 task	3 task	2 task	3 task	**Baseline**
HalfCheetah	**2820 ± 297**	2615 ± 132	2618 ± 362	2514 ± 183	2264 ± 160
Hopper	2535 ± 712	8274 ± 46 490	8080 ± 4704	**10 178 ± 2241**	2241 ± 502
Walker2D	**2214 ± 1254**	963 ± 314	846 ± 286	730 ± 343	547 ± 710
InvPend	874 ± 121	884 ± 20	**885 ± 14**	884 ± 13	883 ± 17
InvDbPend	1438 ± 116	**1531 ± 93**	1348 ± 171	1302 ± 152	1366 ± 179
Ant	2021 ± 722	1899 ± 292	2212 ± 607	2064 ± 386	**2463 ± 208**

D Full Transfer

In Figure 6 learning curves can be seen for the HalfCheetah task where we fully transfer the parameters instead of using FTL. As can be seen, this results in a detrimental overfitting effect where the agent does not seem to be learning.

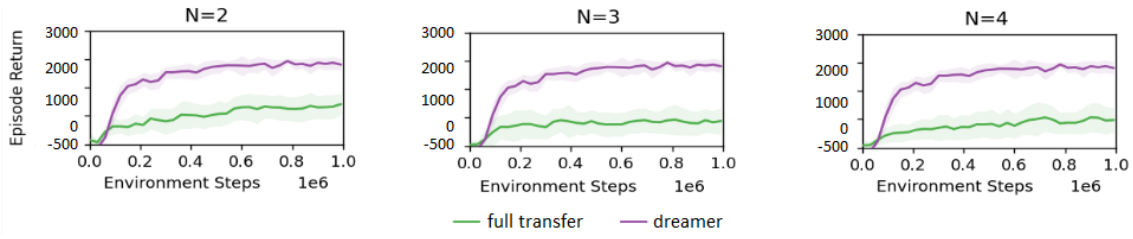

Figure 6: 4 seeds for full transfer learning on the HalfCheetah task across 2, 3, and 4 source tasks.

E Additional Fraction Results

In this appendix, results can be found that illustrate the effect of different fractions α. We present the numerical results of transferring fractions of $\alpha \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]$ using 2, 3, and 4 source tasks. That is, the overall performance over 3 random seeds for the HalfCheetah and Hopper, Walker2D and Ant, and InvertedPendulumSwingup and InvertedDoublePendulumSwingup testing environments can be found in Table 6, Table 7, and Table 8 respectively. Again, as just 3 seeds were used for these experiments, they are not conclusive. However, they do indicate that the choice of α can largely impact performance, and generally, the best performance gains are yielded with $\alpha \in [0.1, 0.3]$.

Table 6: Average return for fraction transfer of 2, 3, and 4 source tasks for the HalfCheetah and Hopper tasks, with fractions $\alpha \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]$. Bold results indicate the best performing fraction per number of source tasks for each target task.

	HalfCheetah			Hopper		
α	2 tasks	3 tasks	4 tasks	2 tasks	3 tasks	4 tasks
0.0	1841 ± 806	1752 ± 783	1742 ± 777	1917 ± 960	1585 ± 941	1263 ± 1113
0.1	**2028 ± 993**	2074 ± 762	1500 ± 781	2041 ± 887	**6542 ± 4469**	3300 ± 3342
0.2	1982 ± 838	1967 ± 862	1773 ± 748	1911 ± 712	5538 ± 4720	1702 ± 1078
0.3	1899 ± 911	**2094 ± 859**	1544 ± 771	**2670 ± 1789**	4925 ± 4695	**3341 ± 4609**
0.4	2008 ± 943	2015 ± 873	**2162 ± 789**	2076 ± 803	2437 ± 2731	1451 ± 991
0.5	1961 ± 944	1647 ± 896	1635 ± 809	1975 ± 772	2014 ± 2328	3246 ± 3828
Baseline		1681 ± 726			1340 ± 1112	

Table 7: Average return for fraction transfer of 2, 3, and 4 source tasks for the Walker2D and Ant tasks with fractions $\alpha \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]$. Bold results indicate the best performing fraction per number of source tasks for each target task.

	Walker2D			Ant		
α	2 tasks	3 tasks	4 tasks	2 tasks	3 tasks	4 tasks
0.0	546 ± 776	**545 ± 1012**	157 ± 891	1070 ± 659	923 ± 559	897 ± 586
0.1	360 ± 803	441 ± 852	**478 ± 1113**	806 ± 469	1022 ± 645	834 ± 652
0.2	**1009 ± 1254**	393 ± 813	200 ± 807	1124 ± 722	1052 ± 687	898 ± 616
0.3	535 ± 914	325 ± 812	323 ± 818	1162 ± 824	1080 ± 701	905 ± 554
0.4	742 ± 992	516 ± 1074	354 ± 862	1308 ± 735	885 ± 571	1022 ± 709
0.5	352 ± 853	355 ± 903	396 ± 829	951 ± 651	932 ± 609	683 ± 498
Baseline		116 ± 885			**1589 ± 771**	

Table 8: Average return for fraction transfer of 2, 3, and 4 source tasks for the InvertedPendulumSwingup and InvertedDoublePendulumSwingup tasks with fractions $\alpha \in [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]$. Bold results indicate the best performing fraction per number of source tasks for each target task.

	InvertedPendulumSwingup			InvertedDoublePendulumSwingup		
α	2 tasks	3 tasks	4 tasks	2 tasks	3 tasks	4 tasks
0.0	847 ± 135	779 ± 259	**838 ± 169**	1255 ± 270	1303 ± 248	1208 ± 322
0.1	857 ± 145	834 ± 198	803 ± 239	1185 ± 316	1283 ± 272	1251 ± 272
0.2	856 ± 121	**850 ± 139**	782 ± 269	1209 ± 280	1299 ± 254	1235 ± 284
0.3	**871 ± 93**	832 ± 182	806 ± 213	1279 ± 216	1325 ± 225	1239 ± 259
0.4	858 ± 150	789 ± 285	790 ± 237	**1307 ± 227**	1323 ± 243	**1278 ± 287**
0.5	852 ± 126	830 ± 181	791 ± 304	1301 ± 216	**1340 ± 230**	1206 ± 278
Baseline		723 ± 364			1194 ± 306	