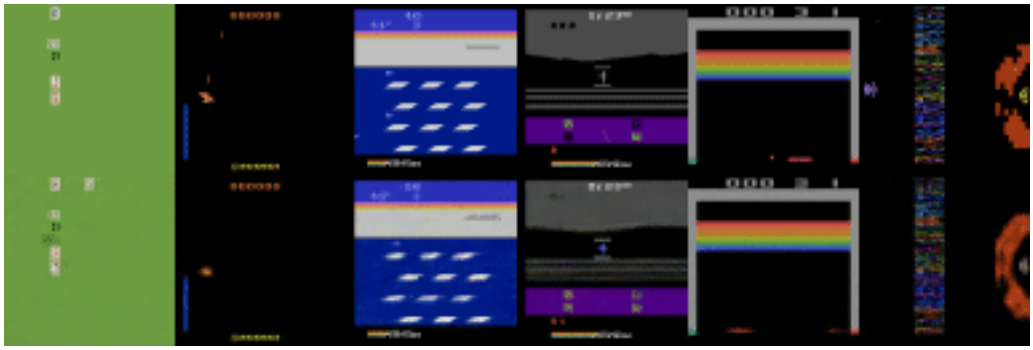
TÉCNICO LISBOA



Efficient Pre-training in Model-based Reinforcement Learning

Bernardo Anjinho Esteves

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Francisco António Chaves Saraiva de Melo

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

June 2022

Acknowledgments

Quero começar por agradecer ao Professor Francisco Melo, que desde o inicio com a bolsa da Gulbenkian, sempre me permitiu seguir a minha curiosidade, que me guiou e ensinou o processo da investigação. Ao Miguel Vasco pelas discussões, artigos e comentários, que sempre conseguiu trazer foco á minha dispersão. Se existe excelência neste trabalho, deve se em grande parte aos vosso apoio.

À fundação Calouste Gulbenkian pela bolsa Novos Talentos em Inteligência Articifial, que deu inicio ao meu percurso de investigação científica, e cujo trabalho e aprendizagens estiveram também refletidas na tese. Um agradecimento ao GAIPS e à RNL pelos recursos computacionais e devido suporte disponibilizados para a realização deste trabalho.

Aos membros do GAIPS e aos meus orientadores pelo feedback dado para a apresentação final de tese. Ao Manuel Goulão e Francisco Lopes pelas discussões e sugestões.

À minha familia, aos meus amigos e aos meus orientadores um obrigado pelo carinho, motivação e conselhos, que tanto nos bons como nos maus momentos, me ajudaram e ajudam a me tornar num humano melhor.

Abstract

Model-based methods and transfer learning approaches, such as pre-training schemes, have both shown promise in addressing the sample-efficiency problem of deep reinforcement learning (RL). In this work, we explore pre-training solutions within model-based frameworks that allow agents to efficiently adapt to novel but similar tasks. We propose a categorization of transferable features between similar tasks that leverages the perceptual, dynamical, and semantic similarities between them. In addition, we propose the Multiple-Augmented Pre-training Scheme (MAPS), a new pre-training algorithm that takes advantage of learning such features in order to build an agent that is more prone to positive transfer, across different environments. We extensively evaluate our approach using a state-of-the-art model-based agent. We perform an ablation study that highlights how the proposed features affect the transfer performance of different components in the architecture of model-based agents. Furthermore, we show how MAPS allows agents to efficiently transfer to similar tasks, outperforming other standard approaches. Finally, by assessing its performance in Atari environments, we show how MAPS easily scales to more complex scenarios.

Keywords

Reinforcement Learning; Transfer Learning; Representation Learning

Resumo

Agentes baseados em modelos e abordagens que usam transferência de conhecimento, tais como esquemas de pré-treino, afiguram-se como métodos promissores para abordar o problema da eficiência amostral dos algoritmos de aprendizagem por reforço profunda. Este trabalho explora métodos de pré-treino aplicados a agentes baseados em modelos de modo a permitir que estes sejam capazes de se adaptarem de forma eficiente a tarefas novas, mas semelhantes. Propõe-se uma categorização de características transferíveis entre tarefas semelhantes, que envolvem semelhanças perceptuais, dinâmicas e semânticas entre elas. É proposto ainda o *Multiple-Augmented Pretraining Scheme* (MAPS), um novo algoritmo de pré-treino que tira partido destas características de modo a aprender um agente transferível entre diferentes ambientes. Avalia-se extensivamente a abordagem usando um agente baseado em modelos de última geração. Realiza-se um estudo de ablação que destaca como as características propostas afetam o desempenho de transferência dos diversos componentes da arquitetura dos agentes baseados em modelos. Além disso, mostra-se como o uso do MAPS permite com que os agentes sejam transferidos de forma eficiente para novas tarefas semelhantes, superando outras abordagens padrão. Por fim, ao testar a performance em ambientes Atari, mostra-se como o MAPS é facilmente escalável para cenários mais complexos.

Palavras Chave

Aprendizagem por Reforço; Aprendizagem por Transferência; Aprendizagem de Representação

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CEM Cross-Entropy-Method

DDPG Deep Deterministic Policy Gradient

MAPS Multiple-Augmented Pre-training Scheme

MDP Markov Decision Processes

MPC Model-Predictive Control

MCTS Monte-Carlo Tree Search

PlaNet Deep Planning Network

PPO Proximal Policy Optimization

RL Reinforcement Learning

RSSM Recurrent State Space Model

SAC Soft Actor-Critic

VAE Variational AutoEncoder

1

Introduction

Contents

From teaching quadrupedal robots to walk on complicated terrains [1], to beating world champions on the game of Go [2], Reinforcement Learning (RL) approaches have been successfully applied to complex scenarios like games [3, 4], robotics [5–7] and control [8]. However, while allowing agents to learn to act in such complex scenarios, RL algorithms are generally sample-inefficient, requiring a significant number of interactions with the environment to learn successfully, and often fail to generalize to similar tasks [9]. This is specially problematic in real world settings, where getting new samples is both slow and expensive, and where the agent might always encounter some situation that it did not encounter during the training phase.

Humans, in contrast, have a learning process that is highly sample-efficient, which reuses prior knowledge of similar tasks (such as motion primitives and environmental physics) to efficiently learn to perform novel tasks [10, 11]. This stark difference motivates the need for transferable knowledge to address the sample-efficiency of RL algorithms.

According to Laskin *et al.* [12], two classes of approaches have been proposed in the literature to address such sample-inefficiency: (i) introducing auxiliary self-supervised tasks on the observations of the agent, in model-free RL methods [12–14]; and (ii) learning predictive world-models in model-based RL methods [11, 15–19]. While the former aims at learning useful representations from the observations of the agent, that can help in the RL task, the latter employs predictive world-models to collect fictitious observations from the environment, thus providing access of the RL agent to "cheap data". Most existing literature follows one of these approaches, for example introducing contrastive and auxiliary losses [12, 14] or do task augmentations [20] to speed up learning, or try to learn a good world-model for fast planning [11, 19, 21]. In this work, we exploit simultaneously the benefits of the two approaches to allow the efficient transfer of knowledge across similar tasks, by addressing the following research question:

> "How to augment the pre-training of model-based RL agents to efficiently fine-tune to novel downstream tasks?"

We focus on the problem of pre-training model-based RL agents to facilitate their transfer to similar tasks. We start by contributing an in-depth categorization of transferable features across similar tasks, where we learn features that capture invariant information between new, similar scenarios that the agent can exploit subsequently in its fine-tuning phase. In particular, we focus our discussion in the transfer between tasks that share perceptual, dynamic and semantic features. Furthermore, we contribute a novel pre-training scheme for model-based RL agents that exploits such transferable features, which we dub Multiple-Augmented Pre-training Scheme (MAPS). During the pre-training phase, MAPS introduces multiple self-supervision tasks based on the observations of the agents obtained in the current or similar tasks, improving the efficiency of the subsequent fine-tuning phase in novel downstream tasks. Such introduction of variability in data has already been explored to fine-tune neural networks to different tasks in contexts such as computer vision [22–24] and natural language processing [25, 26].

We evaluate MAPS against different pre-training approaches in scenarios of increasing complexity, considering a state-of-the-art model-based RL framework (namely, DreamerV2 [19]). We perform an ablation study in a Mini-Grid environment [27] that highlights how changes in the perceptual and dynamical conditions affect the transfer of the components of model-based RL agents to similar tasks. Furthermore, in a more complex Grid-based scenario, we highlight the role of introducing perceptual and semantic variability during the pre-training phase, showing that MAPS outperforms other standard pre-training schemes. Finally, in an Atari environment, we highlight the scalability of MAPS to more complex scenarios, and show how pre-training with MAPS significantly improves the fine-tuning performance.

1.1 Contributions

In summary, the contributions of this work are as follows:

- We contribute a categorization of transferable features for the pre-training of model-based RL agents;
- We introduce the Multiple-Augmented Pre-training Scheme (MAPS), a novel pre-training scheme that exploits such features to introduce variability during pre-training;
- We evaluate MAPS against different pre-training approaches in scenarios of increasing complexity, showing how our approach allows agents to efficiently fine-tune to novel downstream tasks;
- In addition, we propose different grid-based scenarios to evaluate the transfer performance of RL agents and contribute with an open-source Pytorch implementation of DreamerV2[1].

1.2 Outline

This thesis is organized as follows: in Chapter 2, we provide a brief background overview of Reinforcement Learning and Transfer Learning. We finish the chapter with the introduction of the Dreamer agent. We follow in Chapter 3 with a discussion over recent research on model-based RL and Transfer Learning. In Chapter 4 we propose the concept of learnable transfer features and how to build augmentations to learn them, and introduce MAPS. And finally in Chapter 5 we perform an experimental evaluation of the proposed methods. We then conclude with some finals remarks in Chapter 6.

[1] Available at https://github.com/esteveste/dreamerV2-pytorch

2

Background

Contents

2.1 Reinforcement Learning

Reinforcement Learning (RL; [28]) refers to a widely employed computational framework to learn how to solve sequential decision-making problems under uncertainty, through trial-and-error interaction with a given environment. RL problems can be formalized as Markov Decision Processes (MDP) that describe a sequential decision problem under uncertainty [28]. A MDP $\mathcal{M}$ can be instantiated as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$, the state space, corresponds to the set of possible states $s \in \mathcal{S}$ of the environment;
- $\mathcal{A}$, the action space, denotes the set of possible actions $a \in \mathcal{A}$ that the agent is able to perform;
- P, the transition probabilities, describes the dynamics of the environment, where $P(s' \mid s, a) = \mathbb{P}[s_{t+1} = s' \mid s_t = s, a_t = a]$ is the transition function;
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the immediate reward function;
- γ is the discount factor; its value determines how future rewards are valued by the agent.

The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions so as to maximize, for each initial state $s \in \mathcal{S}$,

$$v_\pi(s) = \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \mid s_1 = s, a_t \sim \pi(s_t)\right].$$

In another words, the agent wants to maximize the discounted return, averaged over many episodes. The policy function can be either deterministic or stochastic. If the function is deterministic, we write $a = \pi(s)$ to denote the action in state s. If the policy is stochastic, we write $\pi(a|s)$ to denote the probability of the action a given the state s. This way, $\pi(a|s)$ can be seen as a distribution, where an action sampled from the policy will be denoted as $a \sim \pi(\cdot|s)$.

In this formulation the transition function P only requires the previous state s_t and action a_t to predict the next state s_t. We might consider a different formulation where the probability of the environment transitioning to the state s_{t+1} is conditionally dependent on all of the previous states s and actions a made in the environment. However by being dependent on all previous states and actions, is hard to model the transition function in this form, especially if an episode has many time steps. Thus the presented simplification, where s_{t+1} only depends on the previous state s_t and action a_t becomes useful, on what is known as the *Markov property*. Despite the simplification, this idea is powerful, since any state can be made to include all the necessary information so that the transition function is Markov [29].

One important thing to note is that the agents do not have access to the transition function, $\mathcal{P}(s_{t+1}|s_t, a_t)$, or the reward function, $\mathcal{R}(s_t, a_t, s_{t+1})$. To get the information about these functions, the agent has to interact with the environment, by receiving experience tuples, (s_t, a_t, r_t, s_{t+1}).

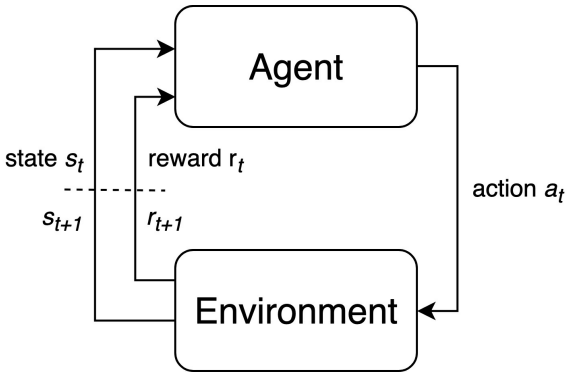


Figure 2.1: The agent-environment interaction in a reinforcement learning setting. From Graesser et al. [29].

Summarizing, in the RL framework, an agent interacts with a (potentially) stochastic environment in a sequence of steps in order to learn how to perform some task. As shown in Fig.2.1, at each-time step:

1. The agent observes the current state s_t of the environment,
2. Given the current state s_t, the agent selects an action a_t following some policy π,
3. The agent's *action* a_t receives the *reward* r_t from the environment which, subsequently changes to *state* s_{t+1}.

We denote the tuple (s_t, a_t, r_t, s_{t+1}) as the experience of the agent at time-step t. The interaction between the agent and the environment can repeat forever, however in practice it ends either by reaching a terminal state, or by reaching a maximum time step $t = T$. The interval between the time step $t = 1$ and the time step when the interaction ends is called an *episode*. A *trajectory* τ is then a possible sequence of experiences within an episode, $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$.

2.1.1 Deep Reinforcement Learning Algorithms

There are three major families of deep reinforcement learning algorithms - policy-based, value-based, and model-based methods, that learn the policy, value, and transition functions. Besides these methods, there are also methods that combine learning more than one of these functions. We can see an overview of some of the most known deep reinforcement learning algorithms from each family represented in Fig. 2.2.

These algorithms can also be classified into *model-based* and *model-free* methods, as delimited on Fig. 2.2.

Contrary to model-free approaches, model-based approaches explicitly learns a model of the environment and uses such model to compute/approximate the optimal policy [30]. Specifically, model-based RL approaches typically learn the different components describing the interaction of the agent with the environment, such as

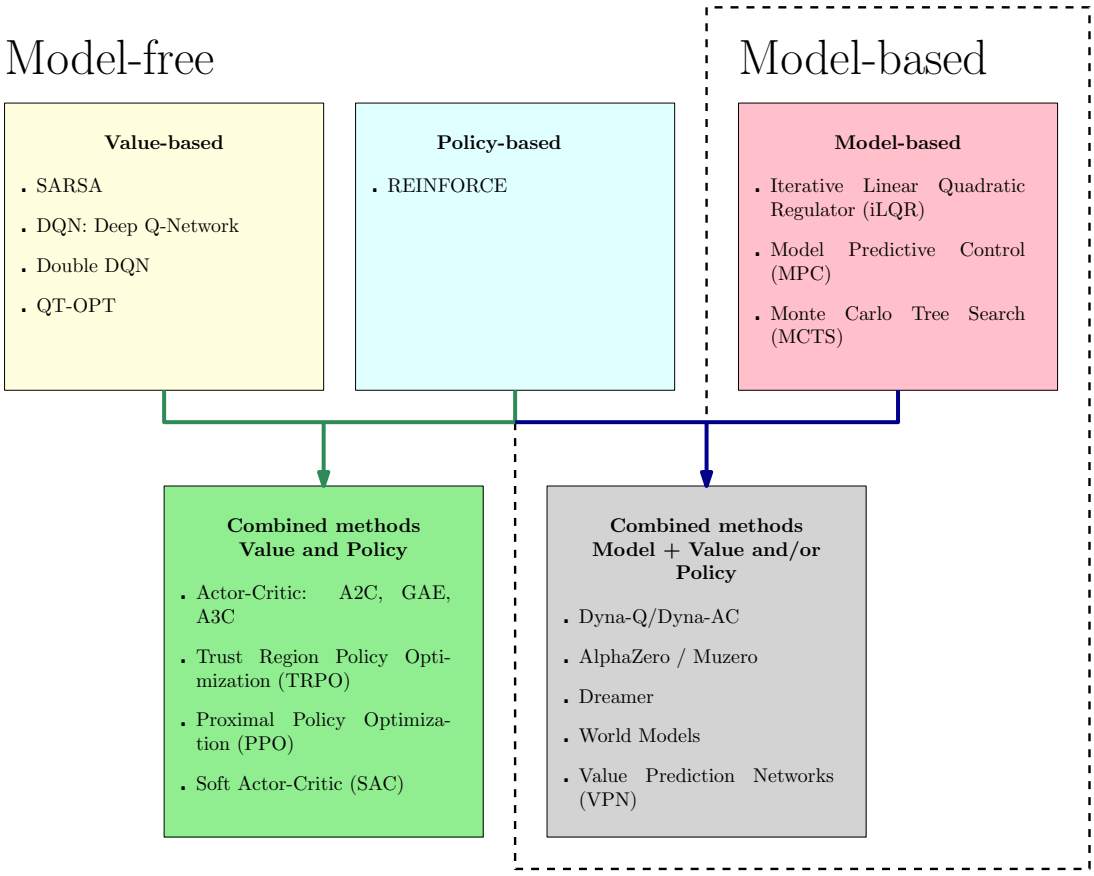


Figure 2.2: Families of deep reinforcement learning algorithms, including some algorithm examples. Inspired in Graesser et al. [29].

- A reward model, $\hat{R}(s) \sim p(r_t \mid s_t = s)$, that predicts/samples a reward r_t as a function of the state s_t. This model can be used to generate synthetic state-reward pairs (s, r).
- A transition model, $\hat{P}(s, a) \sim p(s_t \mid s_{t-1} = s, a_{t-1} = a)$, that predicts/samples a state s_t, given the previous state, s_{t-1}, and the previous action of the agent, a_{t-1}. This model can be used to generate synthetic transitions (s, a, s').

In scenarios where the agent cannot access s_t and must, instead, rely on high-dimensional observations, some models learn an additional representation model, $\hat{S}(o) \sim p(s_t \mid s_{t-1}, a_{t-1}, o_t = o)$. This model is usually recurrent (hence its dependence on s_{t-1} and a_{t-1}) and estimates the current state given the current observation o and the agent's past history. This model is usually used to track the state s_t when the latter cannot be directly observed. Other approaches employ a simpler representation model, $\hat{S}(o) \sim p(s_t \mid o_t = o)$, such as World Models [15], that do not require the agent history. In practice, the representation model is usually trained in an unsupervised manner to learn a compact latent representation s_t of the high-dimensional observation o_t [31]. All these learned functions act as the agent's internal model of the world, and the agent can now compute/approximate the optimal policy by performing planning on its internal model.

2.2 Transfer Learning

Transferring knowledge between tasks is a problem addressed in the field of Transfer Learning, which seeks to bring learning improvements by relaxing the common Deep Learning assumption that the data used for learning between old and new tasks must be independent and identically distributed [32].

Pre-training is considered the predominant approach to perform experience transfer, and in this work we use pre-training as the basis for knowledge transfer [33]. Pre-training requires two sets of tasks, a pre-training task, that we will refer from now on as T_p, and a downstream task, or T_d. Pre-training works as follows:

1. First, we train a model on the pre-training task, T_p;
2. Then, we adapt the model on the downstream task, T_d, by using the previously learned weights, via fine-tuning.

We can also define "component transfer" as the process of training the complete model in the pre-training task and subsequently loading the learned weights of the target component(s) in a new, randomly-initialized model. Additionally, we call fine-tuning as the process of executing the normal RL algorithm in the downstream task after pre-training. In Fig. 2.3, is an example of a model composed by components A,B and C, where we transfer B to learn a new task via fine-tune.

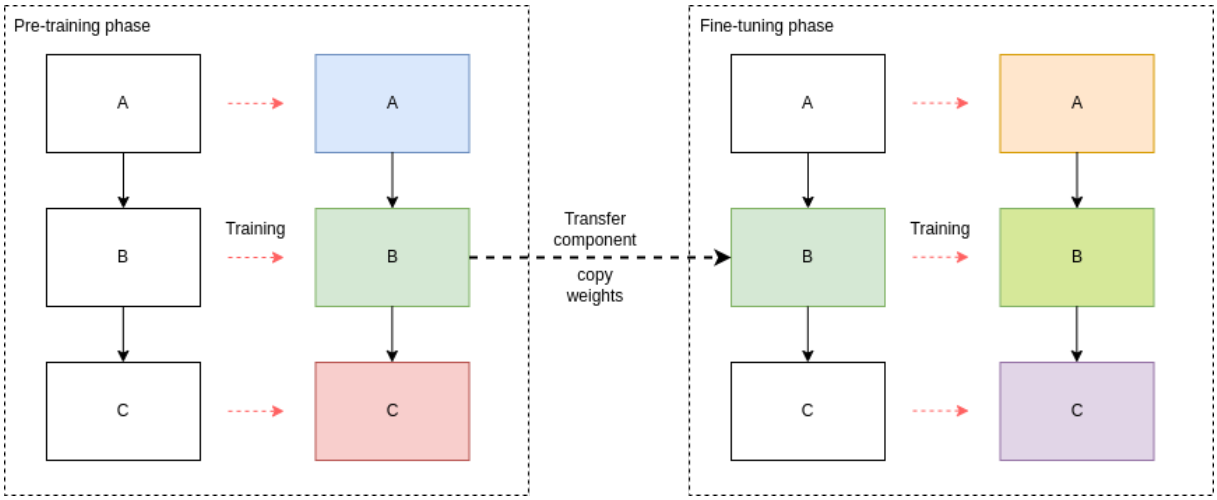


Figure 2.3: Example of transferring the component B to a newly initialized model to be used for learning a downstream task. White squares represent newly initialized component parameters, while colored squares represent parameters that where learned during training, where different colors represent different parameter configurations.

When performing transfer learning, ideally we want the model to be able to use the previously learned knowledge to learn the downstream task faster and achieve equal or better performance than if it was trained without previous information. This idea of a positive contribution in the transfer of previous knowledge in the learning process, is know as positive transfer. However, the opposite event is also possible, where the learned parameters of a previous pre-training task might saturate the network, or learn unuseful functions that need to be unlearned, resulting in a decrease in training efficiency, commonly denominated as a negative transfer.

2.3 Dreamer

In this work, we explore the efficiency of pre-training model-based agents within the framework of DreamerV2 [19], a state-of-the-art model-based agent. DreamerV2 agent builds on several previous works. Therefore, to provide adequate context, we start by introducing PlaNet [34] and DreamerV1 [35], and then present DreamerV2 with an efficient and effective model-based agent for complex visual tasks on discrete action spaces.

2.3.1 PlaNet

Deep Planning Network (PlaNet) is an agent that learns the environment dynamics from images and then performs planning in the latent space of the dynamics model [36]. PlaNet uses a Recurrent State Space Model (RSSM) [34, 37], a latent dynamics model with deterministic and stochastic elements, that

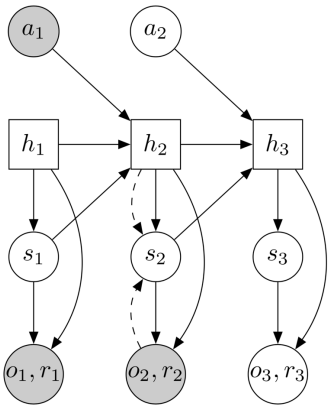


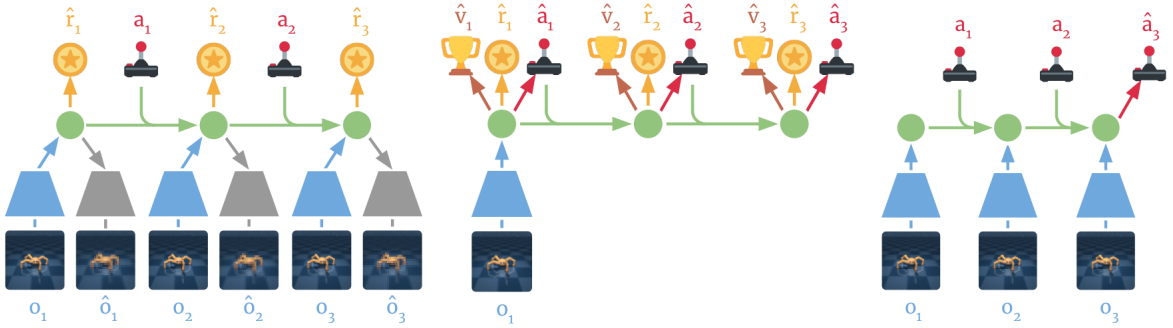
Figure 2.4: Recurrent State-Space Model (RSSM) from the PlaNet paper [36]. In this diagram, the models observe the first two time steps and predict the third. The stochastic variables are represented as circles and the deterministic variables as squares. The solid lines denote the generative process and the dashed lines the inference model.

the authors show to be crucial for successful planning. The RSSM consists of an representation model, a transition model, a observation model and a reward model. These latent model components can be described as follows:

$$\begin{aligned}
\textbf{Recurrent model} &: h_t = f(h_{t-1}, s_{t-1}, a_{t-1}) \\
\textbf{Representation model} &: s_t \sim q(s_t|h_t, o_t) \\
\textbf{Transition predictor} &: \hat{s}_t \sim p(\hat{s}_t|h_t) \\
\textbf{Observation predictor} &: \hat{o}_t \sim p(\hat{o}_t|h_t, s_t) \\
\textbf{Reward predictor} &: \hat{r}_t \sim p(\hat{r}_t|h_t, s_t)
\end{aligned} \tag{2.1}$$

In Fig. 2.4, we further can see how the RSSM model observes two time steps and predicts a third. By following the diagram and (2.1), when training the dynamics model, the recurrent model receives the action a_{t-1}, the deterministic state h_{t-1} and stochastic state s_{t-1} from the previous time step and computes the next deterministic state h_t. Then h_t is used to compute the stochastic state $\hat{s}_t$ with the transitional model. And finally by concatenating both the deterministic h_t and stochastic $\hat{s}_t$ states we can compute the observation $\hat{o}_t$ and reward $\hat{r}_t$ with the observation and reward models, respectively. When receiving a new observation from the environment, with the deterministic h_t and posterior observation o_t, we can get the posterior stochastic state s_t by using the representation Model (represented by the dashed lines).

With this architecture, PlaNet then uses a Model-Predictive Control (MPC) [38] agent to replan at each step. More specifically, it uses for planning the Cross-Entropy-Method (CEM) [39, 40] with the RSSM to search for the best action sequence. Therefore, in contrast to most model-free methods, by using this algorithm no explicit policy or value network is used. With this approach, PlaNet achieves a



(a) Learn dynamics from experience (b) Learn behavior in imagination (c) Act in the environment

Figure 2.5: Training stages of DreamerV1. Image from Hafner et al. [35]

close performance to strong model-free algorithms, on the continuous tasks MuJoco [41] and DeepMind control suite [42]. This method focus however on solving continous control tasks, thus is an unsuitable choice since in this work we decided to focus on discrete control tasks due to its diverse and challeging visual elements provided by common video games.

2.3.2 DreamerV1

Based on the contributions of PlaNet, the Dreamer [35] model was proposed, a novel model-based agent capable of solving long-horizon tasks from learning purely by latent imagination. We can divide the Dreamer agent into 3 steps: dynamics learning, behaviour learning and environment interaction. These 3 algorithmic components are represented in Fig. 2.5.

Like PlaNet, Dreamer uses the recurrent state-space model (RSSM) as world model. Following the previous RSSM explanation and its components in (2.1), Dreamer trains the dynamics model using as loss

$$Loss_d = \mathbb{E}\left[\sum_t \log p(o_t|s_t,h_t) + \log p(r_t|s_t,h_t) - \beta D_{KL}\left[q(s_t|h_t,o_t)||p(s_t|h_t)\right]\right]. \tag{2.2}$$

where β regularizes the information from o_t to s_t. The loss is constituted by 2 reconstruction terms, for observations and rewards, and a KL regularizer. The expectation is then taken over the dataset and representation model.

Dreamer uses an actor-critic approach for behaviour learning, learning an action and a value models defined over the latent space of the world model, i.e.,

$$\begin{aligned} &\textbf{Action model} : a_\tau \sim q_\phi(a_\tau|\hat{s}_\tau,\hat{h}_t) \\ &\textbf{Value model} : v_\psi \approx \mathop{\mathbb{E}}_{q(.|\hat{s}_t,\hat{h}_t)}[\Sigma_{\tau=t}^{t+H}\gamma^{\tau-t}\hat{r}_\tau] \end{aligned} \tag{2.3}$$

The action model implements the agent's policy, predicting the actions that solve the learned environ-

ment, while the value model estimates the expected returns according to what the action model achieves from each state s_t.

Dreamer learns new behaviours starting from some a sequence of model states, $s_{0:t}$, and then generating new "imagined" trajectories $\{s_\tau, a_\tau, r_\tau\}_{\tau=t}^{t+H}$ from its learned model, starting from timestep t until an predefined imagination horizon $t+H$. To properly learn the action and value models, we need to estimate the state values of these imagined trajectories.

Dreamer uses V_λ for value estimation, an explonentially-weighted average of the *k-step* estimates (V_N^k) for different *k* to balance bias and variance,

$$\begin{aligned} V_N^k(s_\tau, h_\tau) &\doteq \mathop{\mathbb{E}}_{q_\theta, q_\phi}[\sum_{n=\tau}^{h-1} \gamma^{n-\tau} r_n + \gamma^{h-\tau} v_\psi(s_h)] \text{ with } h = \min(\tau+k, t+H) \\ V_\lambda(s_\tau) &\doteq (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} V_N^n(s_t, h_t) + \lambda^{H-1} V_N^H(s_t, h_t) \end{aligned} \tag{2.4}$$

where V_λ can also be seen as the λ-return in TD(λ) [28].

The action and value models are trained cooperatively as in policy iteration, where the action model tries to maximize the estimate of the value, and the value model tries to match the estimate of the value that changes with the action model, following

$$\begin{aligned} &\textbf{Action loss} : \max_\phi \mathop{\mathbb{E}}_{q_\theta, q_\phi}[\sum_{\tau=t}^{t+H} V_\lambda[s_\tau, h_\tau)] \\ &\textbf{Value loss} : \min_\psi \mathop{\mathbb{E}}_{q_\theta, q_\phi}[\sum_{\tau=t}^{t+H} \frac{1}{2} ||v_\psi(s_\tau, h_\tau) - V_\lambda(s_\tau, h_\tau)||^2] \end{aligned} \tag{2.5}$$

where θ represents the parameters of the dynamics model, and ϕ, ψ represent the parameters of the action and value models, respectively.

The value model is updated to match the target by following the value loss and the gradient is stoped from propagating at the value model. The action model learns to maximize the value estimates by propagating the gradients from the value and dynamic models, by stochastic backpropagation. To propagate the gradients over distributions, Dreamer uses the reparameterization trick [43] for continuous actions and latent states, and straight-through gradients [44] for discrete actions. Meanwhile, the world model parameters are fixed while learning behaviors. This technique of backpropagating through the value model to learn the action model is similar to Deep Deterministic Policy Gradient (DDPG) [8] and Soft Actor-Critic (SAC) [45].

Finally, Dreamer interacts with the environment by using the stochastic state s_t obtained from the representation model and the environment and performs an action according to the action model, like in the Fig. 2.5. The deterministic state h_t is also computed using the previous stochastic state s_{t-1}

obtained from the environment.

Dreamer achieves a good performance on 20 visual control tasks from the DeepMind control suite [42], surpassing the performance on all the tasks of Planet, and having a very competitive performance with D4PG [46] and A3C [47] with 20x fewer samples. The authors also tried Dreamer on discrete action tasks like the Atari suite [48], but the results were not as strong as other methods.

2.3.3 DreamerV2

In the following year, an improved version of the Dreamer model was introduced, specifically designed for discrete control, often referred as to DreamerV2 [19]. While applying the DreamerV1 agent as a starting point, the authors proposed some changes that turned out to be beneficial for an improved performance of the agent. In a short summary, the most important changes were:

- Using categorical latents with straight-through gradients [44] instead of Gaussian latents in the world model;
- Using REINFORCE [49] for learning the actor instead of using the dynamics backpropagation;
- Added a KL balancing term to seperatly scale the cross entropy of the prior and the posterior in the KL loss;
- Regularize the policy entropy to incentivize exploration in both data collection and in imagination;
- Learning an aditional discount predictor, $\hat{\gamma}_t \sim p(\hat{\gamma}_t|h_t, s_t)$, to estimate when a episode ends when learning a policy in the model's imagination;

The updated dynamics learning loss from (2.2) adds the discount reconstruction term and uses KL balancing (not represented),

$$Loss_d = \mathbb{E}\left[\sum_t \log p(o_t|s_t,h_t) + \log p(r_t|s_t,h_t) + \log p(\gamma_t|s_t,h_t) - \beta D_{KL}\left[q(s_t|h_t,o_t)||p(s_t|h_t)\right]\right]. \quad (2.6)$$

Finally the actor loss is updated from (2.5) to

$$Loss_a = \max_{\phi} \mathop{\mathbb{E}}_{q_\theta, q_\phi}\left(\sum_{\tau=t}^{t+H} \log p_\phi(a_\tau|s_\tau,h_\tau) sg(V_\lambda(s_\tau,h_\tau) - \upsilon_\psi(s_\tau,h_\tau)) - \eta H[a_t|s_t,h_t]\right), \quad (2.7)$$

where the first term is REINFORCE and the second the entropy regularizer, sg represents stop-gradients, and η is a loss scaling parameter.

Proceeding now to the description of the architecture, in Fig. 2.6a is described a flow of the multiple components of the agent. The representation model here is comprised of the encoder and the posterior

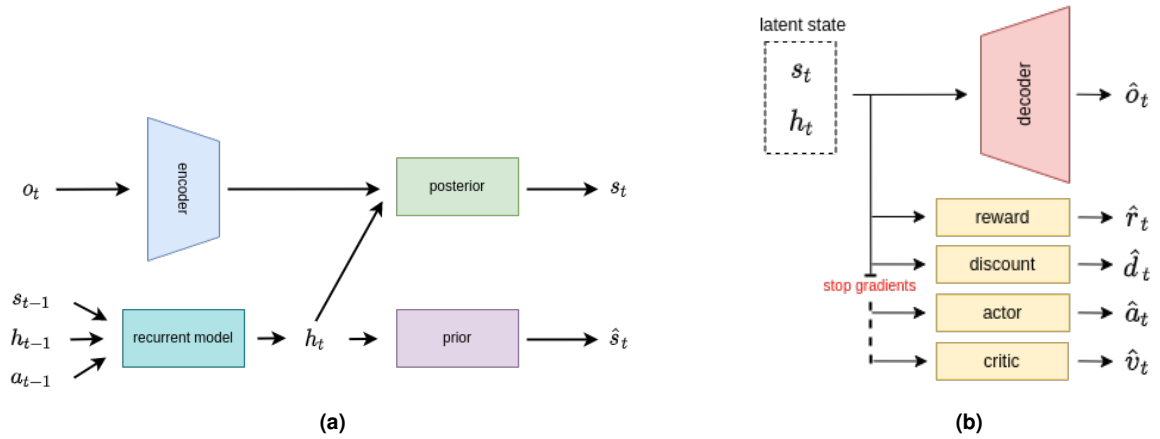


Figure 2.6: The components of the DreamerV2 agent: a) Representation of the flow of the inputs to build the current state through the image observation (representation model $\hat{S}$) and predict the next state (transition model $\hat{P}$). b) Diagram of rest of the components with its multiple objective functions and how the gradient propagates through them

networks that encode the image input o_t into the latent stochastic state s_t. The representation model can also be seen as an image encoder, that in conjunction with the decoder component form a Variational AutoEncoder (VAE) [43]. The transition network is represented by the recurrent model that gives the deterministic state h_t, and the prior network that uses prior information to make a prediction of the current stochastic state $\hat{s}_t$.

The latent state that is made of the concatenation of the deterministic h_t and stochastic s_t states. It is then used in the following networks heads as input; In Dreamer, the decoder of the VAE, the reward predictor and the discount predictor propagate their gradients to build a better internal representation of the state. As denoted in Fig. 2.6b, the gradients of the policy and value functions are not propagated to the rest of the model.

To alleviate the high complexity of modeling a task, Dreamer learns a policy by imagining future time-steps on a short horizon, as well uses as the starting point the latent states used to train the world model from the replay buffer. Following the 3 steps from the previous section and Fig. 2.5, the procedure on how the training of the Dreamer algorithm works is described in Algorithm 2.1 and goes as follows:

0. Lines 1-4: we start by prefilling the replay buffer with some environment samples, using a random policy,

1. Lines 8-10: we perform dynamics learning, by sampling a batch from the replay buffer and training the world model,

2. Lines 12-15: we perform behaviour learning, where the agent learns a policy by imagining new states, using the previous observed states as a starting point,

3. Lines 19-25: the agent interacts with the environment to collect more data samples. And jump back to step 1 until the agent executed the desired number of steps.

Algorithm 2.1: DreamerV2 algorithm in Python pseudo code

```
observation = enviroment.reset()
# we start by filling the replay buffer with some episodes
# using a random policy
prefill_with_random_agent(replay_buffer,enviroment)

while not_finished:
    # train the world model using the episode data from the replay buffer
    batch = replay_buffer.sample_batch()
    wm_loss, latent_states = agent.world_model.train(batch)
    agent.world_model.update_weights(wm_loss)

    # learn to play the policy by imagining new latent states
    # using the replay data as a starting point
    policy_loss = agent.imagine(latent_states,horizon)
    agent.policy.update_weights(policy_loss)

    # for every learning step, Dreamer collects N steps by interacting
    # with the environment
    for _ in range(steps_in_environment):
        action = agent.get_action(observation)
        observation = env.step(action)

        if episode.done(): # save completed episodes to replay buffer
            replay_buffer.save(episode)
            observation = env.reset()
```

By making these changes, the DreamerV2 was able to surpass the performance of other model-free methods like Rainbow [50] or IQN [51] on the Atari 200M benchmark [48], while using the same computational budget and training time.

The interesting phenomena of model-based RL reaching and outperforming top model-free on competitive RL benchmarks, opens new possibilities for efficient transfer and multi-task learning using world-model architectures. Thus we considered DreamerV2 as a good choice for this work.

3

Related Work

Contents

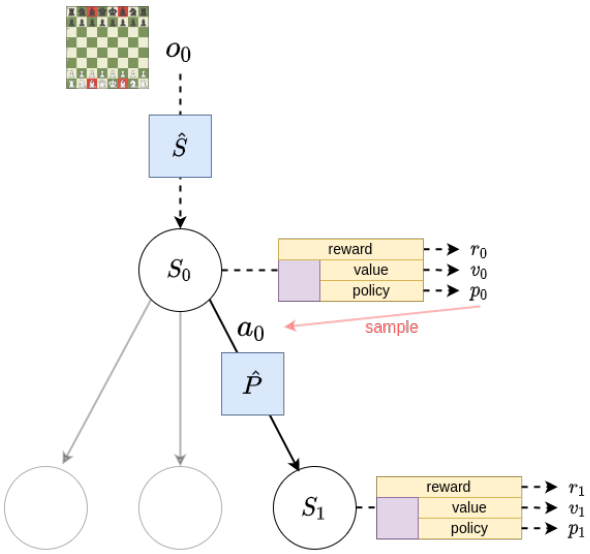


Figure 3.1: Planning in the MuZero architecture, where representation model $\hat{S}$ and the transition model $\hat{P}$ are represented by blue squares, the reward, value and policy networks are represented with yellow rectangles, and where the purple square represents a shared ResNet [52] backbone of the value and policy networks.

3.1 Model-based RL methods

In this section, we present some of the most recent successes in model-based RL research.

3.1.1 MuZero

One of the first big successes in model-based reinforcement learning on high dimensional spaces like images was MuZero [53], spurring a major new enthusiasm in the field. Following the successes of AlphaGo Zero [54] and AlphaZero [55], MuZero learns an unknown sequence model of rewards and values and performs planning of the actions via Monte-Carlo Tree Search (MCTS) [54, 56].

The MuZero agent is composed by the following components:

$$\begin{aligned}
\textbf{Representation model} &: s_t \sim q(s_t|o_t) \\
\textbf{Transition predictor} &: \hat{s}_t \sim p(\hat{s}_t|s_{t-1}, a_{t-1}) \\
\textbf{Reward predictor} &: \hat{r}_t \sim p(\hat{r}_t|s_t) \\
\textbf{Value network} &: \hat{v}_t \sim p(\hat{v}_t|s_t) \\
\textbf{Policy network} &: \hat{p}_t \sim p(\hat{p}_t|s_t)
\end{aligned} \tag{3.1}$$

Following in Fig. 3.1 and the notation presented in Section **??**, MuZero uses a representation model $\hat{S}$ to convert the image inputs o to an initial state s_1, then a Monte-Carlo tree search is performed at each timestep, where it samples an action a_t from the search policy action distributions p_t, and it uses

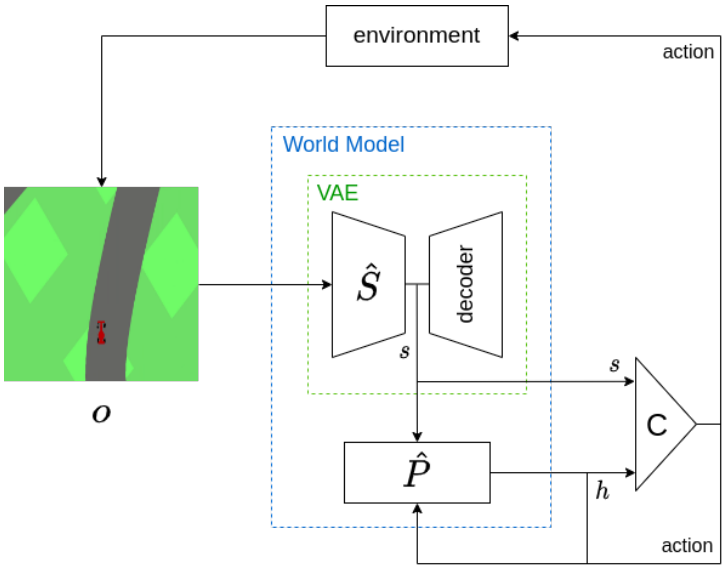


Figure 3.2: The architecture of the World Models agent, modified from [15]. The model comprises a world model and a controller C. The world model, in turn, comprises a representation model ($\hat{S}$) and a transition model ($\hat{P}$).

the transition model $\hat{P}$ to predict the next possible state s_{t+1}.

MuZero creates its sequence model by using only the information of the task reward and does not perform any explicit representation learning using images. With its significant engineering effort, MuZero achieves impressive results on multiple board games and deterministic Atari games.

However, the MuZero agent is not sample efficient, requiring 20 billion frames to train Atari games. Furthermore, the implementation of this algorithm is not available to the public and requires more than 2 months of compution to train one agent on a GPU. In our work, we employ DreamerV2 that, contrary to MuZero, allows for training an Atari agent in a single GPU for 10 days [19].

3.1.2 World Models

Inspired by the human cognition, Ha and Schmidhuber proposed the World Model agent, a model-based framework composed of two key modules (as shown in Fig. 3.2): a representation module $\hat{S}$, that compresses the image input into an internal representation s, and a transition module $\hat{P}$, that corresponds to the agent's internal model of the world's dynamics. With the information from these two modules, a simple controller C is learned to perform actions in the environment.

The representation model is trained using a Variational AutoEncoder (VAE) [43], the structure of which consists of two main parts: an *encoder*, which here is the representation model $\hat{S}$, and a *decoder*. The transition module is composed of a recurrent neural network with a mixture density network (MDN) [57]. These two components, the representation and transition models are considered by the authors as

a *World Model*.

The world model agent is trained sequentially: initially, the agent performs random actions in the desired environment, and collect the image observations and performed actions into a dataset. With that dataset, the authors start by training the VAE to reconstruct the images. Then by fixing the VAE encoder parameters, the observations from the agent are encoded into latent representations which, alongside the actions of the agent, are employed to train the transition model (MDN-RNN) in order to predict the latent representation of the next-time step.

To train the controller, the authors to use evolutionary algorithms, which require large amount of trained data. Thus to minimize the need for large amounts of data, the authors learn the controller entirely inside the world model. However, by doing this, the agent exploits the model's dynamics, and therefore overfits the policy to the model's imperfections. To avoid this, a temperature parameter τ is added to adjust the uncertainty of the model when sampling new states.

Tested on 2 different tasks, the world models agent surpassed the performance of all other referred methods. However, since these tasks are not generally benchmarked, is hard to know how does it compare to with current research. On another note, the authors experimented their model in environments where the full dynamics are observable by performing only random exploration. Thus, more complex environment dynamics require an interactive process, to keep adjusting the world model on the new discovered states. So, even though by training the controller inside the world model might help on the sample efficiency of the genetic algorithm, the iterative process of retraining and finding the ideal temperature value for each iterative step is still very computationally inefficient. Thus, in our work, the choice of DreamerV2 is more computationally and sample efficient since its internal model continuously learns with new data, without the need of finding a good temperature value or number of new environment steps to retrain the world model.

3.1.3 SimPle

Simulated policy learning, or SimPle, uses a video prediction model accompanied by an additional latent variable to directly predicting a future frame based on 4 previous frames [11]. Then, SimPle uses the model-free Proximal Policy Optimization (PPO) [58] algorithm to perform policy optimization inside the simulated frame predictions made by the learned model.

In experimental settings, SimPLe is more sample efficient than the Rainbow algorithm [50] in several Atari games with only 100,000 environment samples. However, with more training steps the benefits in sample efficiency seem to fade away. SimPLe fails to achieve competitive results with the Rainbow algorithm after 500,000 training samples. In contrast, DreamerV2 with more training steps, gets state of the art results, when compared to SimPLe.

For a more complete survey on the field of model-based and deep reinforcement learning we refer

to [31].

3.2 Transfer Learning

Transfer Learning consists of the class of methods that attempt to bring learning improvements by reusing previous learned knowledge. Pan et al. [59] propose three main questions related to transfer learning: "What to transfer", "How to transfer", and "When to transfer".

"What to transfer" looks for which part of the knowledge can be transferred between domains or tasks. Some knowledge is specific for each task, while some other knowledge might be common across different domains and help the performance on the target domain.

Zhu et al. [60], presents a framework for organizing transfer learning methods into 5 different sub-topics: reward shaping, learning from demonstrations, policy transfer, inter-task mapping and representation transfer.

In this work, we focus on representation transfer, where the transferred knowledge are the feature representations, including the representations learned for value or Q-functions. Here we also assume this category includes the transition function in model-based methods.

After setting what knowledge will be transferred, finding the right algorithm to transfer the knowledge corresponds to the "How to transfer" question. "When to transfer" refers to which situations transfer learning should or not be applied. In some situations, if the source and target domains are not related to each other, directly attempting to perform a brute-force transfer might be unsuccessful. Furthermore, it may even hurt the performance of the task domain in a phenomenon called *negative transfer* [61]. Like most research, in this work our main focus is on the first two questions, but taking into consideration how to avoid negative transfer is also an important open issue [59].

3.2.1 How to transfer?

Several different methods have been proposed to address the methodology of how to transfer knowledge between tasks. The most widely-used, pre-training, has been successfully applied to a variety of fields beyond RL. For example, in computer vision, self-supervised representation learning approaches have seen significant developments, both in contrastive [22, 62] and predictive methods [23]. In this work, inspired by those pre-training approaches in computer vision, we explore self-supervised augmentations in the field of model-based RL.

During the pre-training phase, it is common to train the model on large amounts of general data. It is also common to use other learning objectives only during pre-training. For example, in SimCLR [22], the authors present a new contrastive method to pre-train a large model with a large unlabeled dataset with 1.2 million images that can then be fine-tuned with a small labeled dataset. However, in our work we

do not have access to a large dataset with millions of highly diverse trajectories and millions of diverse games easily available. We thus focus the pre-training on a small set of more similar tasks to extract information from these to the desired downstream task.

On the intersection of the areas of representation transfer and model-based RL, there have been multiple successes in performing pre-training and fine-tune on tasks with the same dynamics and different rewards [16,21] and tasks with different dynamics [17,63], and also pre-training on previous collected exploratory/offline-data for learning the task [14, 64].

In RL settings, both CURL [12] and ATC [64] propose contrastive auxiliary objectives for learning general representations of the agent's environments. However, they consider only model-free agents and employ only perceptual augmentations. In this work, we consider how perceptual, dynamical and semantic augmentations improve the transfer of model-based RL agents.

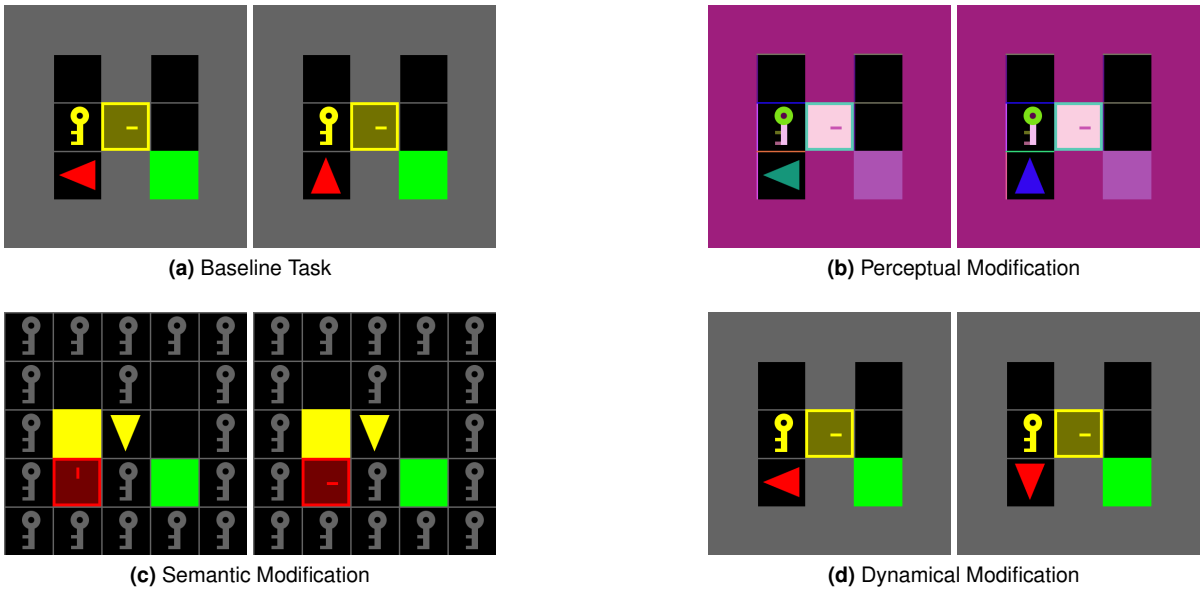
In SGI [14] the authors employ multiple auxiliary tasks to pre-train and then fine-tune an agent on the same task, and show negative results on transferring representations between Atari games on a small data regime. Contrary to our work, they focus only on model-free methods, and only use random crops and intense jittering (both perceptual augmentations). In RAD [20] the authors explore ten different types of data augmentations, and show how using augmentations while learning the same task helps improve the data-efficiency and generalization of RL methods. Compared with our work, RAD uses only perceptual augmentations and focuses on model-free single task learning.

In conclusion, numerous studies show that humans learn new tasks better and faster by transferring the knowledge learned from solving similar tasks [65, 66]. By looking at research, we notice the importance of data augmentations and auxiliary losses to help build better representations more suitable for good transfers. Thus, in this work, we take inspiration from the use of different augmentation methods, to categorize them in 3 different categories, and try to explore how the use of different types of augmentations help on experience transfer.

4

Transferring features in model-based RL

Contents



(a) Baseline Task

(b) Perceptual Modification

(c) Semantic Modification

(d) Dynamical Modification

Figure 4.1: The proposed categorization of transferable features: the similarity between a pre-training task T_p and a downstream task T_d can be interpreted through Perceptual (P), Dynamical (D) and Semantic (S) invariances.

In this work, we address the problem of pre-training model-based RL agents to allow the efficient fine-tuning in novel downstream tasks. A model-based agent interacts with an environment T_p during the pre-training phase, collecting information such as the observations $o_{p,1:H} = \{o_{p,1}, \dots, o_{p,H}\}$, the actions performed by the agent $a_{p,1:H} = \{a_{p,1}, \dots, a_{p,H}\}$ and the rewards received, $r_{p,1:H} = \{r_{p,1}, \dots, r_{p,H}\}$, up to a temporal horizon H. The agent then uses such information to learn to act in the environment, learning the representation, transition and reward functions, $\hat{S}$, $\hat{P}$ and $\hat{R}$, as described in Chapter 2. We investigate how the model $(\hat{S}, \hat{P}, \hat{R})$, learned in a given task, can be used when the agent faces a similar downstream task, T_d.

Such learning process raises the question of what features, computed from $\mathbf{o}_p$, $\mathbf{a}_p$, and $\mathbf{r}_p$, should be transferred from the pre-trained task T_p to the downstream task T_d. We propose to categorize such potentially transferable features according to three classes: perceptual, dynamical and semantic features.

In this chapter, we formalize the properties of these features and contribute with a novel pre-training scheme, MAPS, that exploits such features by building augmentations of the observations of the agent, which in turn facilitate its adaptation to novel similar environments. In the follow sections we start by presenting an intuition on how each class can categorize similar tasks, and then we formalize how models that are able to act in different tasks from the class should be able to represent similar features.

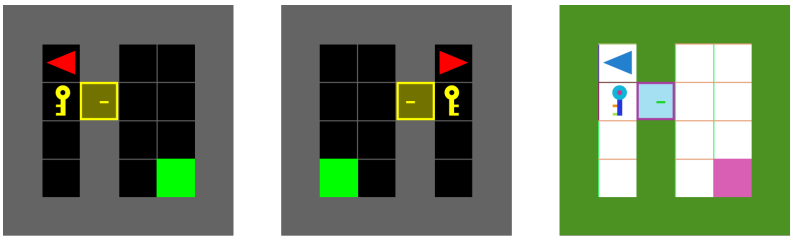


Figure 4.2: Perceptual augmentations: different image observations corresponding to the same state of the MiniGrid environment.

4.1 Perceptual Features

We consider transferable perceptual features as representations shared between different tasks that are encoded only from the observations of the agent. Such features are encoded despite potential differences in the observations, such as visual augmentations of the complete image (flip, resize, inverting colors, etc...). Consider the example provided in given in Fig. 4.2: the original observation (left) can be augmented using transformations such as flipping the image, or inverting the colors.

More precisely, if we consider that two tasks T_a and T_b are able to share perceptual features, we assume that exists a augmentation function A of the observation such that the task T_a can be transformed into the task T_b, i.e., $o_b = A(o_a)$. These tasks are then considered to share perceptual features, if we can learn an agent model such that

$$\hat{S}(o_a) \approx \hat{S}(A(o_a)).$$

As such, the fine-tuning learning performance of the agent in the novel task should be improved, if the novel task shares the perceptual features with the model, learned from previous tasks. The representation model should be able to represent the same environmental state from the different observations provided alone, despite their perceptual differences. This approach has been explored by several self-supervision methods, such as SimClr [22] and CURL [12], that learn transferable representations by employing visual-based augmentations on image data.

4.2 Semantic Features

We consider transferable semantic features as representations shared between different tasks that have the same high-level meaning, such as the player and the objects. Contrary to perceptual features, which only rely on the observations of the agent, these features require information provided from the previous sequence of observations, actions and rewards obtained from the environment. Consider the example provided in given in Fig. 4.3: despite their perceptual differences, the high-level semantic components of the state of the environment in both the original and modified MiniGrid scenario remain the same: there exists a player, surrounded by walls, which requires to grab a key to open a door, even if the objects are

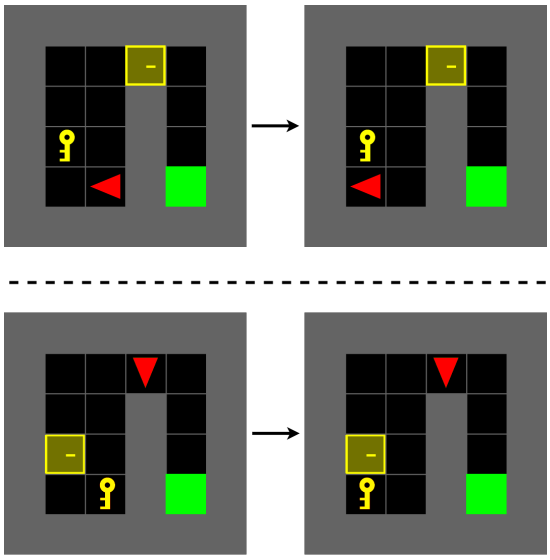


Figure 4.3: Semantic augmentations: two observations from an original and modified MiniGrid environment, highlighting the change in the high-level elements of the task: the player (Up: red triangle, Down: yellow key), the walls (equal: gray squares), the key (Up: yellow key, Down: yellow door), the door (Up: yellow door, Down: red triangle) and goal (equal: green square).

different and the sprites are swapped. To understand the functionality of each semantic components, the agent needs to interact with the environment and remember how the components work.

More precisely, we if have 2 tasks T_a and T_b, from which the task T_b can be constructed from transforming the observations of the task T_a, such that $o_b = A(o_a)$, we can say that these tasks share semantic features if the agent can learn a representation function such that

$$\hat{S}(o_a, h_a) \approx \hat{S}(A(o_a), h_b),$$

where h_k represents the history of the previous state and dynamic information up to the current time-step in task k, i.e. $h_k = \{s_{k,1:t-1}, a_{k,1:t-1}\}$. In this sense, perceptual features are a subset of the semantical features. The main difference, is that for the agent to be able to find the similar state representations between tasks with semantic similarities, it needs to know the history from the previous interactions with the environment. Learning such high-level semantic knowledge suitable for RL task remains a challenging problem [67].

4.3 Dynamical Features

We consider transferable dynamical features as the transition dynamics shared between different tasks such that if we choose the appropriate actions we can make both tasks achieve similar states. Consider the example provided in given in Fig. 4.4, where three different MiniGrid task sequences are represented. All tasks start in the same state, and have different action meanings and dynamics. We can easily

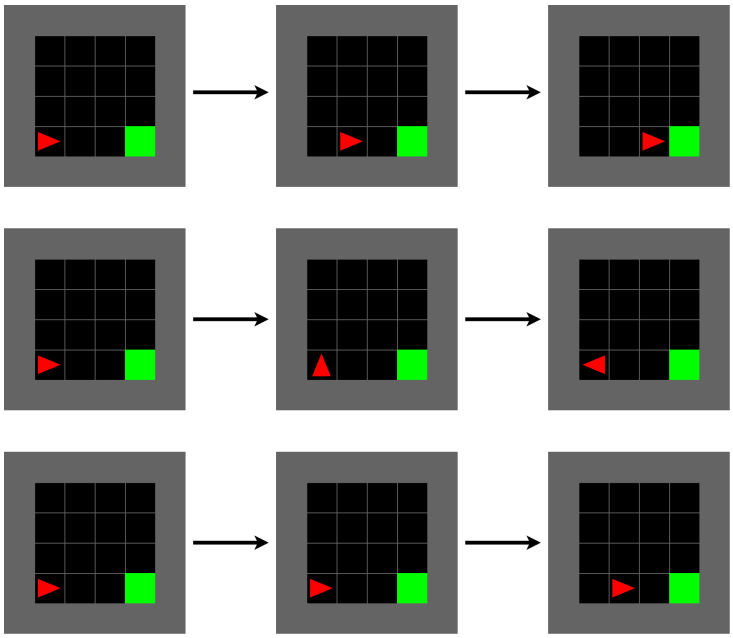


Figure 4.4: Dynamical augmentations: three different observation and action sequences from different time steps t, in modified versions of the MiniGrid environment for action a = forward: original task (top); task with swapped actions, "Forward" $\rightarrow$ "Rotate Left" (middle); task with random NoOp actions (bottom).

achieve the same sequence on the original task, if we make a "Rotate left" action instead the up action for the second sequence, and if we make a NoOp action in the right moment for the third sequence. For simplicity, we always consider the tasks to have the same state space, but this idea could be extended to the intuition that even with different states, different tasks could share similar state transitions.

More precisely, if have 2 tasks T_a and T_b, from which the task T_b can be constructed by transforming the actions of the task T_a, such that $a_b = A(a_a)$, we can say that these tasks share dynamical features if the agent can learn a transition function such that

$$\hat{P}(s_a, a_a, h_a) \approx \hat{P}(s_a, A(a_a), h_b),$$

where h_k represents the history of the previous state and dynamic information in task k up to the current time-step, i.e., the history for the task T_a at time-step t is $h_{a,t} = \{s_{a,1:t-1}, a_{a,1:t-1}\}$ and for the task T_b is $h_{b,t} = \{s_{b,1:t-1}, A(a_{a,1:t-1})\}$. In this sense, with this equation, we expect that for state-action pairs $\{s_{a,n:t}, a_{a,n:t}\}$ and $\{s_{b,n:t}, a_{b,n:t}\}$, if $A(a_{a,n:t}) = a_{b,n:t}$ and $s_{a,n} = s_{b,n}$, then all the following states should be equal, i.e., $s_{a,n:t} = s_{b,n:t}$.

An example of how this formalization of dynamical features could be extended, is to consider a robotic setting of learning a policy in a simulator and transferring the learned policy to the real world, in a problem known as sim2real. Small differences between the real world and the simulator can make transferring the policy fail in the real world, so one thing we can do is to consider slightly different dynamical augmentations to the multiple parameters of the robot control in the simulator, such that the

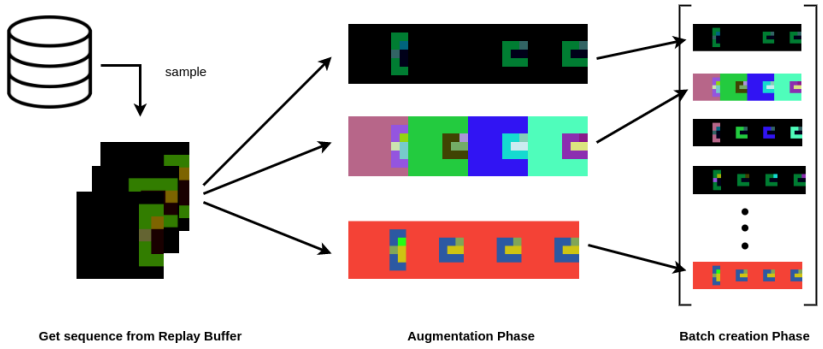


Figure 4.5: The Multiple-Augmented Pre-training Scheme (MAPS) for efficient transfer of model-based RL agent to novel similar tasks: initially, we collect a sequence of observations (using, for example, a random policy); subsequently, we augment that specific sequence with a user-defined transformation; finally, we stack the multiple trajectories into a single training batch.

agent can learn to adapt to the slight different real world control idiosyncrasies [68].

Given these forms of augmentations, we now describe MAPS as an approach that uses combinations of these classes to improve the transfer learning benefits.

4.4 Multiple-augmented pre-training of model-based RL agents

Motivated by recent approaches in self-supervised visual learning [12, 22], we follow the hypothesis that by training the world-model of the agent on augmented versions of the pre-training task, T_p, we force the model to learn features that are transferable to a task T_d requiring only fine-tuning. This is the core idea behind the Multiple-Augmented Pre-training Scheme (MAPS).

MAPS introduces augmentations that exploit the observations of the agent to increase their variability in the perceptual, dynamical, and semantic dimensions, in order to improve the transfer efficiency of the learned internal model of the agent to novel, similar tasks. In order to employ MAPS to pre-train the world-model of the agent, we construct training batches as shown in Fig. 4.5. A training batch is constructed with multiple trajectories of the agent in the pre-training task, T_p, each with its own augmentation. We follow a three-stage approach, as also shown in Algorithm 4.1: initially, we collect previously a sequence of observations (using, for example, a random policy as shown in lines 3-5) and sample a sequence from the replay buffer (line 11); subsequently, we augment that specific sequence with a user-defined augmentation function (line 12); finally, we stack the multiple trajectories into a single training batch (line 13). Note that semantic augmentations can be created by modifying the observations of the agent in the pre-training task, but also by introducing additional similar tasks.

The use of augmentations has been explored in works like Curl [12], RAD [20] and SGI [14] for data efficiency. However, these works focus mainly on perceptual augmentations like flips, color jitter, translations and cutouts. MAPS differs from this research since it focuses on improving transfer efficiency

Algorithm 4.1: Multiple-Augmented Pre-training Scheme (MAPS) pseudo-code

```
    # fill the replay buffer with pre_train task samples,
    # obtained using a random policy
    pre_train_environments = [...]
    for env in pre_train_environments:
        pre_fill_with_random_agent(replay_buffer,env)

    pre_train_agent = Agent()
    for step in range(pre_training_steps):
        batch=[] # create a new training batch
        for i in range(batch_size):
            task, sequence = replay_buffer.sample_task_sequence()
            sequence = augment(sequence,task) # perform selected augmentation
            batch+=[sequence] #create the batch

        pre_train_agent.world_model.train(batch)
```

on learning new tasks and uses perceptual, dynamical and semantic augmentations.

For improving generalization in RL, domain randomization [7, 69, 70] methods have been proposed to transfer learned agents from simulation to the real world. Domain randomization creates a multitude of different tasks by randomizing the proprieties of the simulator like, the lighting settings or colors of the objects (a perceptual augmentation), the simulator physics (a dynamical augmentation) or the number, shape or position of objects in the scene (a semantic augmentation). MAPS extends this idea, by proposing a novel a framework from general and useful augmentations. It learns to generalize on a pre-training task with multiple augmentations, and then performs transfer to a new task, by re-using the previously learned representations to learn the new task faster. It differs from domain randomization since its fine-tuning task might have completely different goals, or certain different representations or states not represented on the pre-training phase. Instead of just transferring successfully the policy with few fine-tuning steps, MAPS serves to speed up the learning process, even if certain elements of the downstream task are completely different.

5

Evaluation

Contents

We evaluate MAPS against other standard pre-training schemes in scenarios of increasing complexity, showing how our approach allows pre-trained model-based RL agents to efficiently transfer to novel, similar tasks. We start by performing an ablation study that highlights how changes in the perceptual, dynamical and semantic conditions affect the transfer of the components of model-based RL agents to similar tasks. We then investigate the role of introducing perceptual and semantic variability during the pre-training phase, showing that MAPS outperforms other standard pre-training schemes.

5.1 Experimental Setup

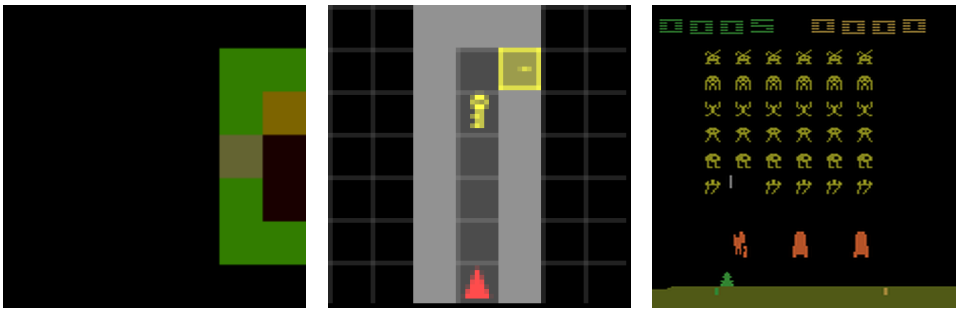
To fully exploit the perceptual, dynamical and semantic variability within the MAPS framework, we consider two different grid-based scenarios in our evaluation:

- MicroGrid: A small-scale 5×5 grid-world environment based on MiniGrid (Fig. 5.1a);
- MacroGrid: A larger-scale 8×8 grid-world environment based on MiniGrid, where the visual observations of the agents are upscaled to 64×64 pixels (Fig. 5.1b).

Both scenarios allow for fine control over the elements of the environment (such as colors, shapes and grid sizes), facilitating the creation of the necessary perceptual and dynamical augmentations for MAPS. In addition, the higher-resolution MacroGrid scenario allows to exploit semantic variability by changing the object sprites present in the environment. In both scenarios, we consider the DoorKey navigation task, which requires that the agent obtains a key to unlock the door that allows it to reach the goal.

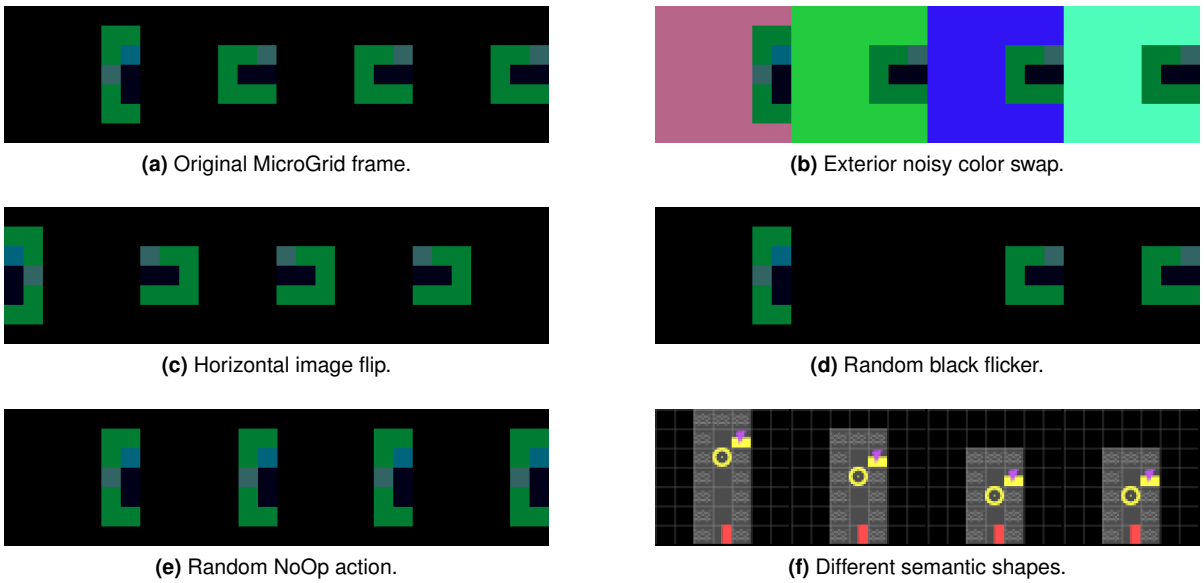
We instantiate the following classes of augmentations for MAPS in the grid-based scenarios:

- Perceptual (P):
  - Static color changes
  - Color changes on every step
  - Spatial visual changes
- Dynamical (D):
  - Modifications that do not change optimal policy: Random NoOp action
  - Modifications that alter optimal policy: Swap actions, Random Any action
- Semantic (S):
  - Image occlusions
  - Swap object sprites positions (only in MacroGrid)
  - Use different object sprites (only in MacroGrid)



(a) MicroGrid. **(b)** MacroGrid **(c)** Atari

Figure 5.1: The environments employed in the evaluation of MAPS.



(a) Original MicroGrid frame.

(b) Exterior noisy color swap.

(c) Horizontal image flip.

(d) Random black flicker.

(e) Random NoOp action.

(f) Different semantic shapes.

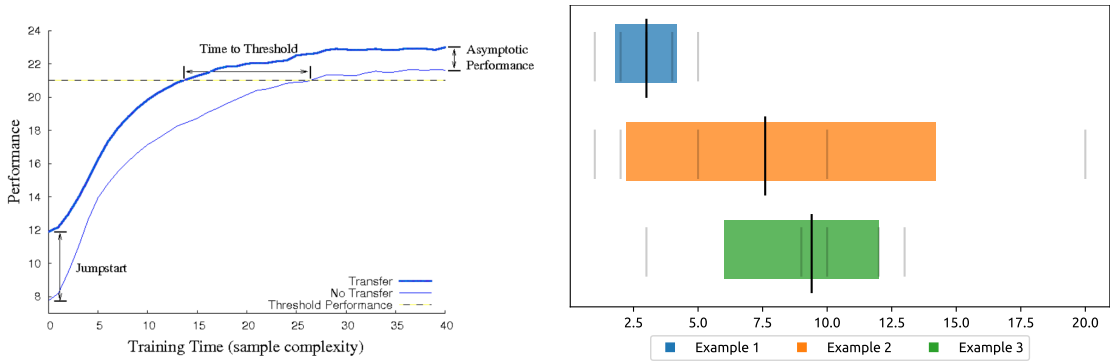
Figure 5.2: Augmentations employed in MAPS for the Mini-Grid environment.

We consider 5 different augmentations as modified tasks that are used throughout all experiments, represented in Fig. 5.2. We consider two perceptual augmentations (exterior noisy color swap and horizontal image flip), one dynamic augmentation (random NoOp action) and two semantic tasks (random black flicker, MacroGrid with semantic data).

Finally, we also test the MAPS framework in the Atari game environment [48]. Inspired by the grid-based results, we also create general augmentations applicable to any Image-based game. An Environment frame are shown in Fig. 5.1c.

We evaluate the sample-efficiency of MAPS by following the metrics presented in [71]: an algorithm is more sample efficient than another if it reaches a higher performance in the same training window. If the algorithms present similar asymptotic performances, then we compare the jump-start performance and area under the curve. An example of how to interpret these metrics is present in Fig. 5.3a.

For visualizing and comparing the different areas under the curves we follow the recommendation



(a) (b)

Figure 5.3: Evaluation metrics of transfer learning performance: (a) Representative graph from Taylor et al. [71] that shows the multiple metrics presented by the author to compare transfer learning approaches in the same plot: jump-start, asymptotic performance, time to threshold and total reward (area under the learning curve); (b) Example of an interval estimate of 3 different tasks with 5 runs each with stratified bootstrap confidence intervals. The dark line represents the mean value and gray lines the seeds of the task. The colored interval represents the estimate where 95% of the data is expected to lie. Shaded cells are observed, white cells are estimated. Best viewed with zoom.

of [72] to perform an interval estimate via stratified bootstrap confidence intervals. However, to perform bootstrap it is recommended to have at least 20 seeds per task [73], something that is not feasible due to computational restraints and time limitations. Therefore, we proceed to visualize an interval estimate using regular summary statistics. An example of how to interpret these plots is in Fig. 5.3b, where we can assert that Example 3 has better performance than Example 1, for at least 95% of the times, and that while we cannot affirm that Example 2 has better performance than Example 1, we can say they have comparable performances, even though the mean of Example 2 is higher.

We also note that none of the data used in the pre-training phase, including the data generated by the augmentations, is ever seen in the fine-tuning phase. We instantiate our pre-training scheme considering the DreamerV2 agent [19], discussed in Section 2.3.

5.2 Ablation Study

We start by performing an initial ablation study in order to understand the general contribution of each component of the model-based agent, to the overall transfer process, without MAPS. We pre-train the agent in the MicroGrid scenario for 225k time steps and subsequently transfer selected components to the same task ($T_p = T_d$), with the remaining components having their weights randomly initialized. We consider that the model is composed by a representation model ($\hat{S}$), a transition model ($\hat{P}$) and a reward model ($\hat{R}$), thus referring to the complete world model as $\hat{S} + \hat{P} + \hat{R}$.

We evaluate the sample efficiency of the transferred agents in learning a new policy in comparison with a baseline of learning both the world model and policy from scratch, as shown in Fig. 5.4. The results

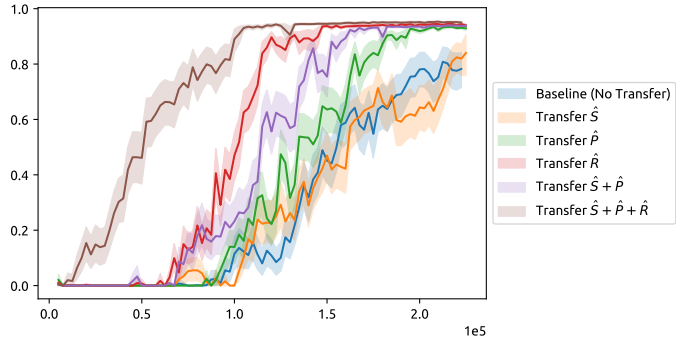
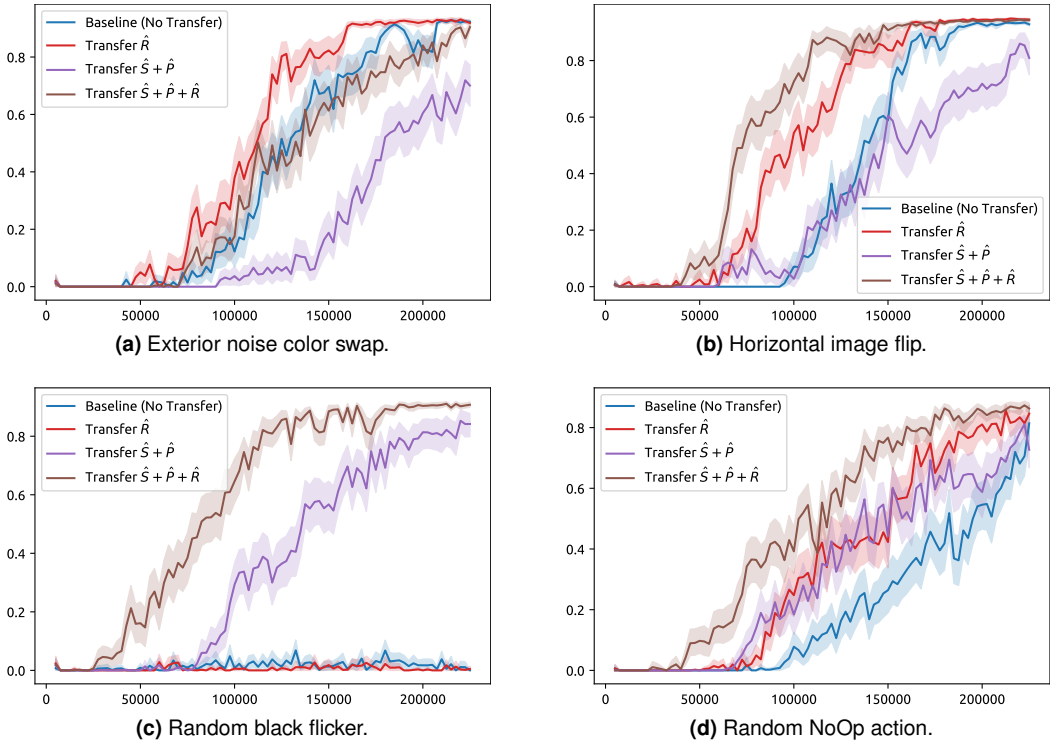


Figure 5.4: Transfer performance of selected components of pretrained agents in MicroGrid to the same task ($T_p = T_d$). Results averaged over 10 randomly-seeded runs.



(a) Exterior noise color swap.

(b) Horizontal image flip.

(c) Random black flicker.

(d) Random NoOp action.

Figure 5.5: Transfer performance of selected components of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

show that transferring the world model with all components ($\hat{S} + \hat{P} + \hat{R}$) offers a substantial speedup to an agent when learning a newly initialized policy. Furthermore, transferring the reward predictor ($\hat{R}$) appear to have the biggest contribution in transfer. Our experiments show that the highest contribution of this component comes from the good prediction of when no reward is given, where by successfully providing a prediction of zero, it allows the agent to better explore the environment without exploiting the noisy process of learning the sparse rewards.

On the contrary, transferring only the representation model ($\hat{S}$) does not provide a significant contribution over the baseline, only originating a positive contribution when transferred in conjunction with

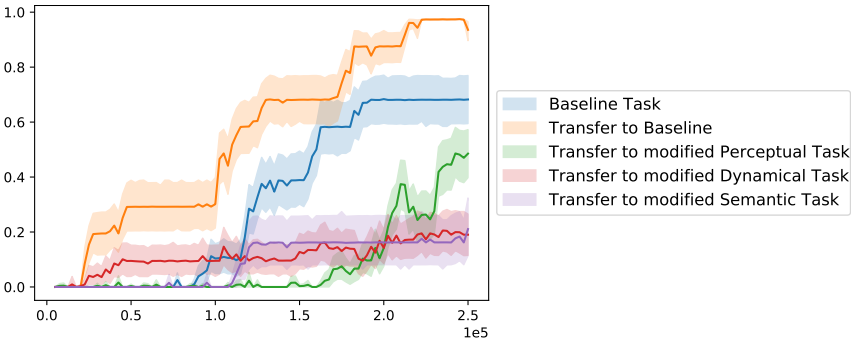


Figure 5.6: Transfer performance of fine-tuning the model-based world model, pre-trained on a baseline task, to modified versions of the baseline with different perceptual, dynamical and semantic features

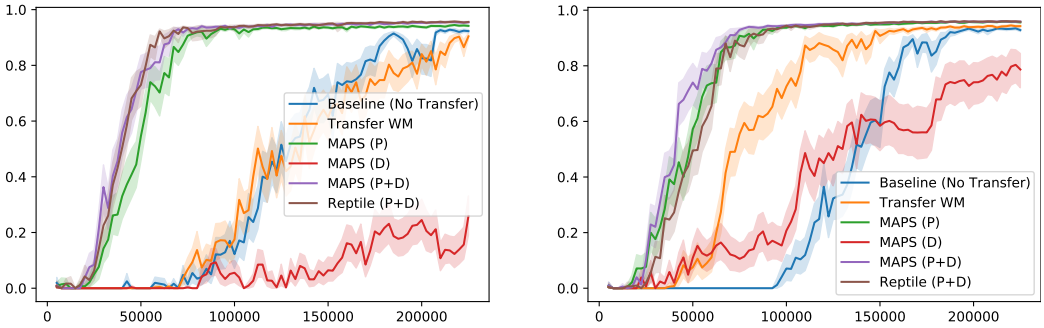
the transition model ($\hat{S} + \hat{P}$). This is due to the fact that, in the Dreamer architecture, the representation and transition models are coupled and, as such, in general the transfer process must also consider model-specific and architecture-specific constraints.

Overall, the results show that the transfer performance of model-based agents is constrained by the successful transfer of the representation ($\hat{S} + \hat{P}$) and reward models ($\hat{R}$). We thus perform the same ablation study in scenarios of transfer to similar tasks ($T_p \neq T_d$), without MAPS: we pre-train the agent in the MicroGrid scenario for 225k time steps and subsequently transfer selected components to one of the augmented versions of MicroGrid, discussed in Section 5.1.

We evaluate the sample efficiency of the transferred agents in learning a new policy in comparison with a baseline of learning both the world model and policy from scratch, as shown in Fig. 5.5. The results show, once again, that transferring the complete world model ($\hat{S} + \hat{P} + \hat{R}$) allows agents to more efficiently learn a policy in the downstream scenario, resulting in positive transfer. However, for some augmentations, transferring the representation model ($\hat{S} + \hat{P}$) results in a negative transfer, as seen in Fig. 5.5a, due to the significant differences in the observations provided by the environment. This result motivates the need to introduce variability in the pre-training phase by employing, MAPS. We also note that in scenarios where information about the current state is often omitted, such as in Fig. 5.5d, the baseline agent is unable to learn the task from scratch and only the agent with transferred world model is able to adapt to this new task. This result further motivates the need for transferring the components of pre-trained agents to novel, challenging tasks.

5.3 Pre-training of Model-based RL Agents

We propose to explore transferable perceptual, dynamical and semantic features in the pre-training of model-based RL agents to improve the efficiency of their subsequent fine-tuning in novel downstream tasks. A naive approach to explore this problem would be to pre-train a model-based agent on a specific task, and then fine-tune it in a downstream task with different transferable features. However with this



(a) Exterior noise color swap. **(b)** Horizontal image flip.

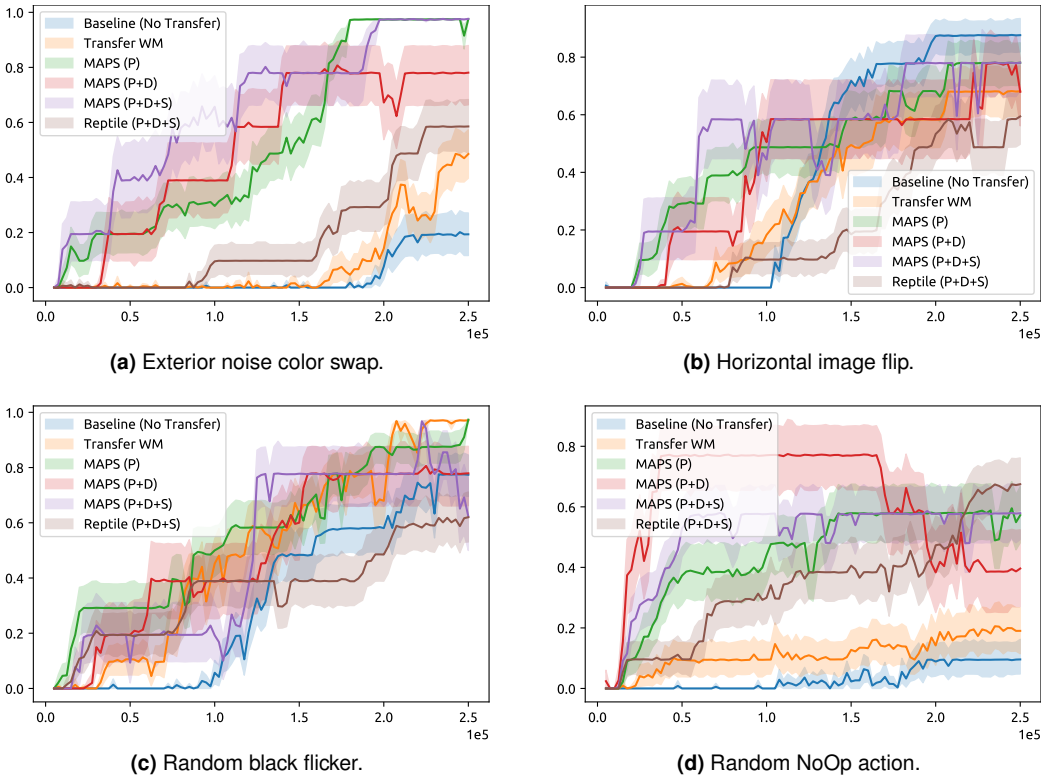
(c) Random black flicker. **(d)** Random NoOp action.

Figure 5.7: Transfer performance of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

approach, the agent can just overfit and memorize specific representations to solve the task, resulting in the learning of features that might not be transferable to downstream tasks with different representations. This would lead to a decrease in training efficiency and thus in a negative transfer. This phenomena can be observed in Fig. 5.6, where we transfer the world model pre-trained on the baseline task, to different tasks.

Therefore we introduce MAPS in the pre-training phase for a successful transfer of model-based agents. We consider two transfer scenarios: we pre-train the agent in the MicroGrid or MacroGrid scenario for 225k or 250k time steps, respectively, and subsequently transfer the world model to an augmented version of the pre-trained scenario, following the augmentations in Section 5.1. We compare the fine-tuning performance of the agents that were pre-trained with MAPS against: agents pre-trained without MAPS and agents without pre-training (learning from scratch). Furthermore, we also compare the MAPS approach to a meta learning, as meta learning as been successfully employed for transfer learning. As such we employ Reptile [74], a state-of-the-art first-order algorithm, as a baseline.

For these environments, we attempt multiple augmentation settings to better ascertain how the increase of pre-training variability can help transfer. We start by attempting single augmentations in perceptual (P) and dynamical (D) categories, and then move on attempting to use all the available combinations of augmentations like perceptual-dynamical (P+D) and perceptual-dynamical-semantic (P+D+S).



(a) Exterior noise color swap.

(b) Horizontal image flip.

(c) Random black flicker.

(d) Random NoOp action.

Figure 5.8: Transfer performance of pretrained agents in MacroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

For further details on the used pre-training augmentations, we refer to Appendix B.2.

We present the results for the MicroGrid scenario in Fig. 5.7. Overall, the results show that MAPS has a significant contribution to a positive transfer to the downstream task. This improvement is clearly seen for the Exterior noisy color swap downstream tasks, where introducing perceptual augmentations with MAPS during pre-training allows the agent to efficient adapt to the downstream task. The improvement can be extended to all other augmentations, where transferring the world model pre-trained with MAPS results either in a higher asymptotic performance or/and a better jump-start learning performance. The

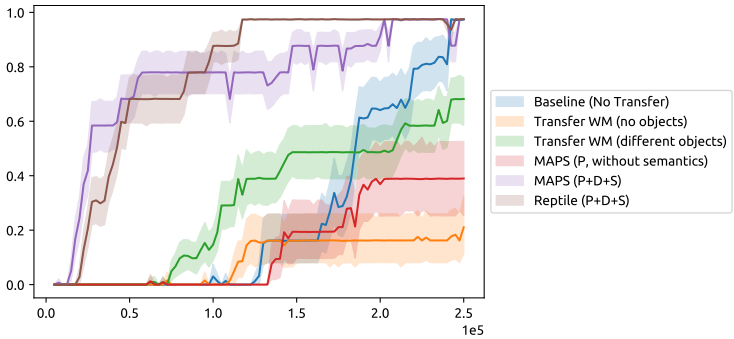
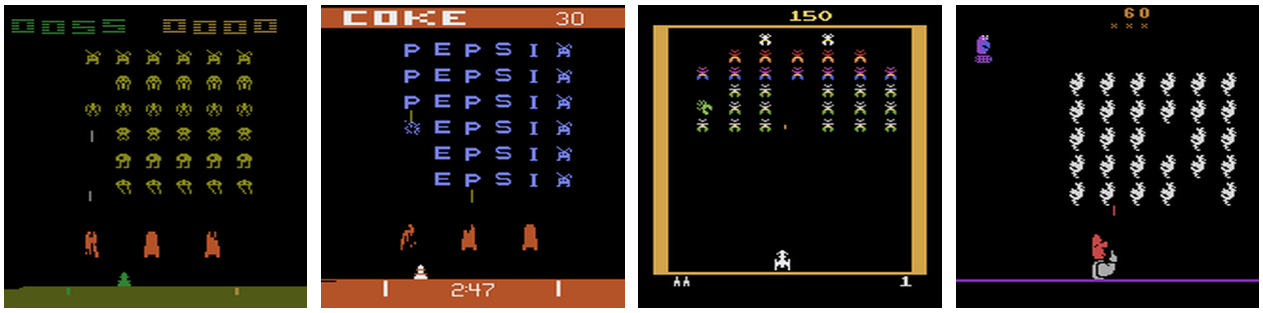


Figure 5.9: Transfer performance of pretrained agents in MacroGrid to a semantically-augmented task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.



(a) Space Invaders **(b)** Pepsi Invaders **(c)** Galaxian **(d)** Pigs in Space

Figure 5.10: Similar tasks considered for the Atari scenario: all games have the player moving in straight line, shooting up enemies aligned in a squared grid formation.

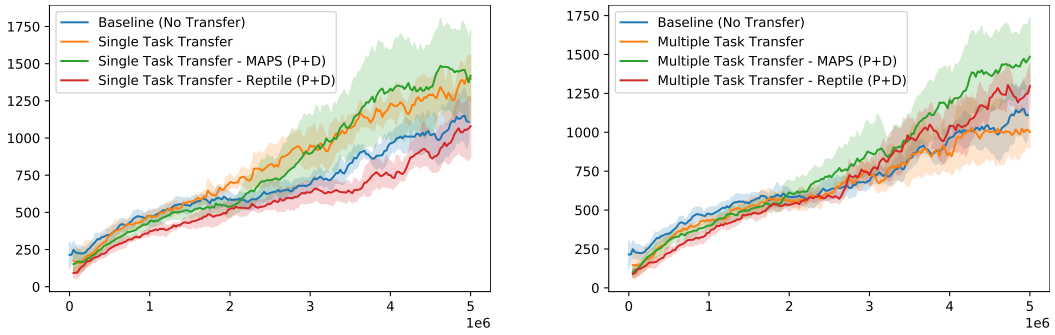
same results are valid when comparing MAPS with Reptile, where MAPS always has a higher asymptotic performance or/and a better jump-start learning performance.

We verify a similar trend in the results for the MacroGrid scenario, presented in Fig. 5.8. The results show, once again, that transferring the world model pre-trained with MAPS results overall in a higher asymptotic fine-tuning performance or/and a better jump-start fine-tuning performance. Moreover, in the MacroGrid scenario we can also evaluate the transfer to tasks with distinct semantic features (such as when changing the sprites of the objects in the environment). We present such results in Fig. 5.9: only the agent that pre-trains with MAPS, considering perceptual and semantic augmentations, is able to positively transfer to this challenging task, with a significant jump-start fine-tuning performance over the baselines. The results also show that MAPS outperforms or has similar performance to the meta-learning approach of Reptile, thus showing that joint-training of tasks can still be a strong alternative over meta-learning methods. Overall, the results attest to the importance of introducing variability regarding perceptual, dynamical and semantic features using MAPS during the pre-training phase to a positive transfer of model-based RL agents.

5.4 Atari Games

Finally, to understand how the performance of MAPS scales to more complex scenarios, we evaluate our approach in Atari Games [48]. In this complex scenarios we both pre-train and fine-tune all agents up to 5M steps.

For this environment we considered general perceptual and dynamical augmentations, as shown in Section 4.4, which can be employed in any image-based scenario, regardless of its complexity. Furthermore, due to the intrinsic difficulty of creating semantic augmentations in complex scenarios, we explore the use of similar tasks in order to exploit semantic features during the pre-training of MAPS. Therefore, for this section, we created and use a set of general perceptual, dynamical and semantical



(a) Single Task Transfer **(b)** Multiple Task Transfer

Figure 5.11: Transfer performance of pretrained agents in (a) Pepsi and (b) 3 similar task to Space Invaders. Results averaged over 5 randomly-seeded runs.

augmentations for any image based environments that we refer only as MAPS. Further details on the augmentations can be found in Appendix B.2.

We consider two different training scenarios: initially, we select Pepsi Invaders as the pre-training task and transfer to Space Invaders, to evaluate the role of perceptual and dynamical augmentations in the performance of MAPS, a scenario we denote by Single Task Transfer. Secondly, we select a group of three similar games (Pepsi Invaders, Galaxian, Pigs in Space) as pre-training tasks, to exploit semantic variability during pre-training, and transfer the agent to Space Invaders, a scenario we denote by Multiple Task Transfer. The selected pre-training tasks share the same grid like structure of the enemies as the fine-tuning task, as shown in Fig. 5.10.

We present our results in Fig. 5.11. In the Single Task Transfer we verify that pre-training on the similar Pepsi task yields a positive improvement, while pre-training jointly on Multiple Task Transfer has a negligible to negative performance over training from scratch. Using MAPS brings a significant improvement over the Single Task pre-training, making it the best performing method for training the Space Invaders tasks with 5M time-steps when compared with other publicly available results in Table 5.1. In the Multiple Task Transfer scenario (Fig. 5.11b), the results show once again that employing MAPS

Table 5.1: Comparison of the final score at step 5M in Space Invaders.

Method	5M Avg. Score
DQN [75]	808 ± 38
C51 [76]	1035 ± 30
Rainbow [50]	1086 ± 81
IQN [51]	**1602** ± **153**
DreamerV2 [19]	1141 ± 295
Single Task Transfer	1393 ± 466
Single Task Transfer - MAPS (P+D)	**2032** ± **774**
Multiple Task Transfer	863 ± 335
Multiple Task Transfer - MAPS (P+D)	**1813** ± **576**
Single Task Transfer - Reptile (P+D)	1226 ± 458
Multiple Task Transfer - Reptile (P+D)	1719 ± 483

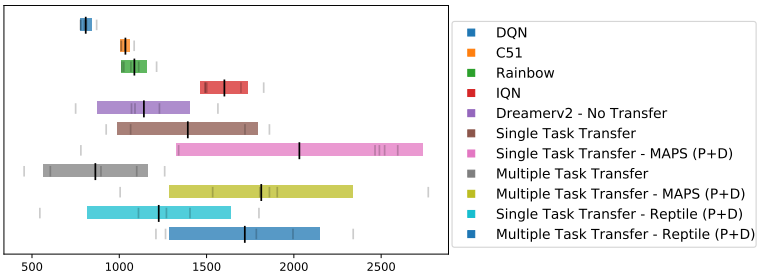


Figure 5.12: Final transfer performance of pretrained agents against other learning algorithms (without transfer) at step 5M on the game Space Invaders.

allows for a significant positive transfer over the baseline and over the naive pre-training approach.

Is worth mentioning that while both pre-training methods with MAPS have a high mean, both also have a big variance in the results, as seen in Fig. 5.12. This higher variance in the performance results also seems to be a characteristic of the DreamerV2 method. Thus, we can ascertain that our methods using MAPS are better with a 95% CI than DQN [75], C51 [76] and Rainbow [50], while being competitive with IQN [51] and baseline DreamerV2 [19]. On another hand, we can also conclude that learning using Multiple Task Transfer with MAPS is significantly better than without.

6

Conclusions

Contents

In this work, we investigated the introduction of perceptual, dynamical and semantic variability during the pre-training of model-based RL agents for an efficient transfer of the agents to novel tasks. We introduced the concept of similar perceptual, dynamical or semantic features between different tasks, and created a formal framework to build augmentations that help the agent better generalize over such features. We contributed with MAPS, a novel pre-training scheme that introduces augmentations over the observations of the agent to take advantage of such variability. Our results show that MAPS improves the fine-tuning efficiency of pre-trained agents to novel downstream tasks of increasing difficulty.

We started by defining the properties of the information shared between different tasks in the transfer process, which we denote by transferable features. We have approached such definition through perceptual, dynamical and semantic lenses, defining where such features arise in the architecture of standard model-based RL approaches. Additionally, we have proposed how to leverage such features during the pre-training phase of the agents, contributing with specific augmentations in grid-based and Atari environments. We have provided a novel Pytorch implementation of the DreamerV2 agent that successfully replicates the original work in the Tensorflow framework.

We have addressed the question of which components of model-based RL architectures play a fundamental role in the transfer procedure. Our results have shown that, while the reward component plays a significant role, the transfer of all components are fundamental to maximize the fine-tuning performance of the agent. We evaluated MAPS against other baselines (both with and without pre-training and transfer) in grid-based settings, showing that MAPS outperforms the other baselines. We have shown that employing only perceptual augmentations, as done in the majority of literature, may not be enough for a positive transfer. Using MAPS with a complete set of perceptual, dynamical and semantic augmentations was shown to be the best approach to reach a positive transfer on any similar task. We also compared MAPS with a meta-learning approach, and shown no improvement for using a meta-learning algorithm over the joint-training method for learning the augmentations with MAPS. Finally, we have shown that MAPS improves the learning performance of agents in Atari games scenarios, outperforming other state-of-the-art model-free algorithms with the same number of environment steps.

With this work we make a small step towards the creation of sample-efficient agents, that can efficiently re-use its knowledge, to continuously learn how to perform new tasks. Our method also made a small step towards the concept of a general world model representation, where a single instance of an agent pre-trained with MAPS, could be used to improve the learning performance on multiple different MiniGrid tasks. Like with the many successes in computer vision [22–24] and natural language processing [25, 26], we believe that pre-training will play a major role in the development of a foundational world model [33] that can be used as a starting point for learning any new reinforcement learning task quickly.

6.1 Future Work

Previous work has shown the learning and transfer benefits arising from scaling up both the model architectures and the training data [33]. Thus attempting to explore how bigger models are able to learn more general representations, capable of obtaining positive transfers across more broad sets of tasks, is an interesting research path. Regarding the DreamerV2 agent, using highly expressive and more recent network architectures like ResNet [52] or image transformers [77, 78] is also a good approach. In the same line of thought, Gato [79] builds a general agent that is trained using imitation learning on several different tasks and games. An interesting continuation of this idea, is to try to learn a large world model on a large amount of different reinforcement tasks, and try to see if it is possible to learn a general world model that can learn new tasks faster. The diverse pre-training scheme introduced in MAPS is fundamental for the goal of learning a general and flexible world-model, that can lead to a positive transfer across various tasks.

Another approach, would be to test MAPS on different model-based algorithms or different auxiliary losses like contrastive learning methods. While DreamerV2 uses image reconstruction to build its internal representations, recent works on model-based with contrastive methods like EfficientZero [80] show to be very efficient on learning meaningful representations with very few data. Doing an analysis of MAPS on this, is an interesting research path to verify which models and losses are better for learning a good internal representation that is good for transfer learning. Related to this, it is interesting as well to test this approach on model-free algorithms. While research methods like RAD [20] show how augmentations can help to learn a model-free agent policy, it would be interesting to see how can the use of augmentations help on better generalizing transfer learning to new tasks. Using model-free algorithms pre-trained with augmentations on multiple-different tasks are also an interesting research path, for the search of a general model-free representation model or agent. The use of a RL objective as a single learning objective for building an internal representation is an interesting approach since it tries to learn only the important features for solving the task, and thus if trained on many different tasks and augmentations, it might be able to learn a general representation that is very efficient.

Another interesting research path, might be in performing an analysis on how pre-trained task policies or pre-trained exploratory policies might further improve the transfer benefits. This research path is also related to field of multi-task reinforcement learning.

Finally, another open possibility is to explore how MAPS with offline data can bootstrap world-model representations to quickly learn new tasks like on the Atari 100k benchmark [11]. It could be interesting to see how augmentations can help to build better representations on small amounts of data collected by random or exploratory policies.

Bibliography

[1] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[2] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *CoRR*, vol. abs/1504.00702, 2016.

[6] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *CoRR*, vol. abs/1603.02199, 2016.

[7] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," 2019.

[8] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[9] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artificial Intelligence*, 2018, pp. 3207–3214.

[10] B. Lake, T. Ullman, J. Tenenbaum, and S. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, 2017.

[11] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model-based reinforcement learning for Atari," *CoRR*, vol. abs/1903.00374, 2019.

[12] M. Laskin, A. Srinivas, and P. Abbeel, "CURL: Contrastive unsupervised representations for reinforcement learning," in *Proc. 37th Int. Conf. Machine Learning*, 2020, pp. 5639–5650.

[13] M. Jaderberg, W. Czarnecki, I. Dunning, L. Marris, G. Lever, A. Castaneda, C. Beattie, N. Rabinowitz, A. Morcos, A. Ruderman *et al.*, "Human-level performance in 3D multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019.

[14] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, R. Hjelm, P. Bachman, and A. Courville, "Pretraining representations for data-efficient reinforcement learning," in *Adv. Neural Information Proc. Systems 34*, 2021.

[15] D. Ha and J. Schmidhuber, "World models," *CoRR*, vol. abs/1803.10122, 2018.

[16] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," *arXiv preprint arXiv:1907.01657*, 2019.

[17] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Proc. 2018 Conf. Robot Learning*, 2018, pp. 617–629.

[18] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," *CoRR*, vol. abs/2005.05960, 2020.

[19] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Mastering Atari with discrete world models," *CoRR*, vol. abs/2010.02193, 2020.

[20] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," in *Adv. Neural Information Proc. Systems 33*, 2020, pp. 19 884–19 895.

[21] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," *CoRR*, vol. abs/2005.05960, 2020.

[22] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. 37th Int. Conf. Machine Learning*, 2020, pp. 1597–1607.

[23] J. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Pires, Z. Guo, M. Azar *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," in *Adv. Neural Information Proc. Systems 33*, 2020, pp. 21 271–21 284.

[24] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proc. IEEE/CVF Int. Conf. Computer Vision*, 2021, pp. 9650–9660.

[25] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[26] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[27] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

[28] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[29] L. Graesser, W. Keng, and a. O. M. C. Safari, *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, ser. Addison-Wesley data and analytics series. Addison-Wesley Professional, 2019. [Online]. Available: https://books.google.pt/books?id=V-JJzQEACAAJ

[30] T. Moerland, J. Broekens, and C. Jonker, "Model-based reinforcement learning: A survey," *CoRR*, vol. abs/2006.16712, 2020.

[31] A. Plaat, W. Kosters, and M. Preuss, "Model-based deep reinforcement learning for high-dimensional problems: A survey," *CoRR*, vol. abs/2008.05598, 2020.

[32] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proc. Int. Conf. Artificial Neural Networks*, 2018, pp. 270–279.

[33] R. Bommasani, D. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *CoRR*, vol. abs/2108.07258, 2021.

[34] M. Karl, M. Soelch, J. Bayer, and P. Van der Smagt, "Deep variational bayes filters: Unsupervised learning of state space models from raw data," *arXiv preprint arXiv:1605.06432*, 2016.

[35] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[36] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2555–2565.

[37] A. Doerr, C. Daniel, M. Schiegg, N.-T. Duy, S. Schaal, M. Toussaint, and T. Sebastian, "Probabilistic recurrent state-space models," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1280–1289.

[38] A. G. Richards, "Robust constrained model predictive control," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[39] R. Y. Rubinstein, "Optimization of computer simulation models with rare events," *European Journal of Operational Research*, vol. 99, no. 1, pp. 89–112, 1997.

[40] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *arXiv preprint arXiv:1805.12114*, 2018.

[41] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[42] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, "Deepmind control suite," 2018.

[43] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[44] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[45] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1861–1870.

[46] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.

[47] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[48] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artificial Intelligence Res.*, vol. 47, pp. 253–279, 2013.

[49] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[50] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[51] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 1096–1105.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[53] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[54] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[55] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[56] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[57] C. M. Bishop, "Mixture density networks," 1994.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[59] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[60] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *CoRR*, vol. abs/2009.07888, 2020.

[61] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, "Characterizing and avoiding negative transfer," 2019.

[62] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[63] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema networks: Zero-shot transfer with a generative causal model of intuitive physics," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1809–1818.

[64] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling representation learning from reinforcement learning," in *Proc. 38th Int. Conf. Machine Learning*, 2021, pp. 9870–9879.

[65] R. S. Woodworth and E. Thorndike, "The influence of improvement in one mental function upon the efficiency of other functions.(i)." *Psychological review*, vol. 8, no. 3, p. 247, 1901.

[66] D. N. Perkins, G. Salomon *et al.*, "Transfer of learning," *International encyclopedia of education*, vol. 2, pp. 6452–6457, 1992.

[67] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart *et al.*, "Relational deep reinforcement learning," *CoRR*, vol. abs/1806.01830, 2018.

[68] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 737–744.

[69] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[70] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[71] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Machine Learning Res.*, vol. 10, no. 7, 2009.

[72] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[73] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "How many random seeds? statistical power analysis in deep reinforcement learning experiments," *arXiv preprint arXiv:1806.08295*, 2018.

[74] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[75] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[76] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 449–458.

[77] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[78] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[79] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg *et al.*, "A generalist agent," *arXiv preprint arXiv:2205.06175*, 2022.

[80] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, "Mastering atari games with limited data," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[81] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

A

Evaluation Results

A.1 Ablation Study

In Fig. A.1 and Fig. A.2 are represented the ablation results on 12 different MicroGrid tasks, with the performance and area under the curve interval plots, respectively.

A.2 Pre-training of Model-based RL Agents

In Fig. A.3 and Fig. A.4 are represented the complete transfer performance comparison on 12 different MicroGrid tasks, with the performance and area under the curve interval plots, respectively. Followed by the MacroGrid results in Fig. A.5 and Fig. A.6, for MAPS with perceptual (P) or dynamical (D) augmentations.

A.3 Zero-shot reconstruction on new tasks

We also attempted to see how would a pre-trained world model behave if it received a new task that it had not seen before. In Fig. A.7 and Fig. A.8 we explore how pre-training the world model in Single and Multiple Task Transfer, respectively, is able to represent the Space Invaders game. As we can notice in Fig. A.7b and Fig. A.8b, using MAPS appears to help in the reconstruction of the new unseen task.

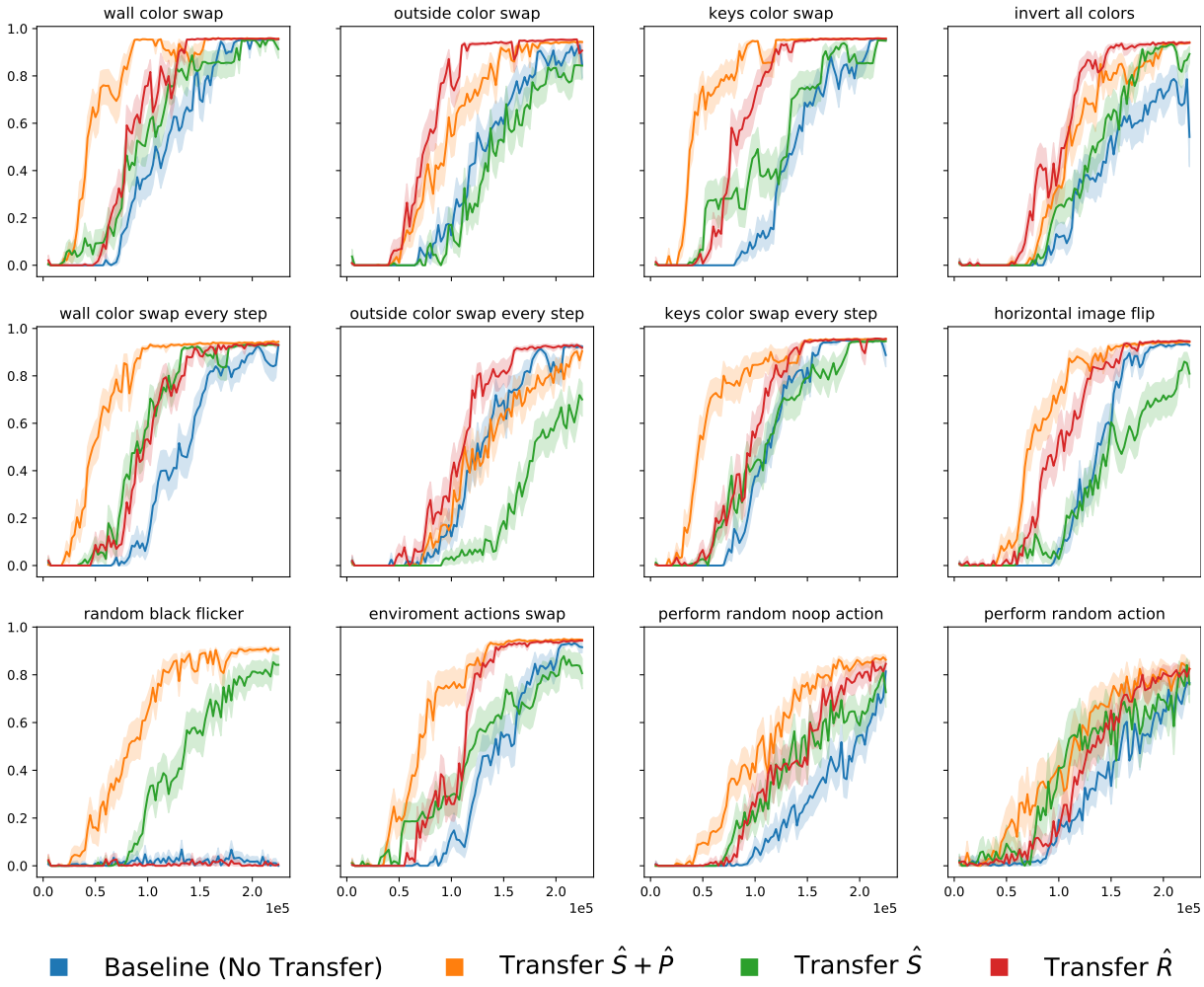


Figure A.1: Transfer performance of selected components of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

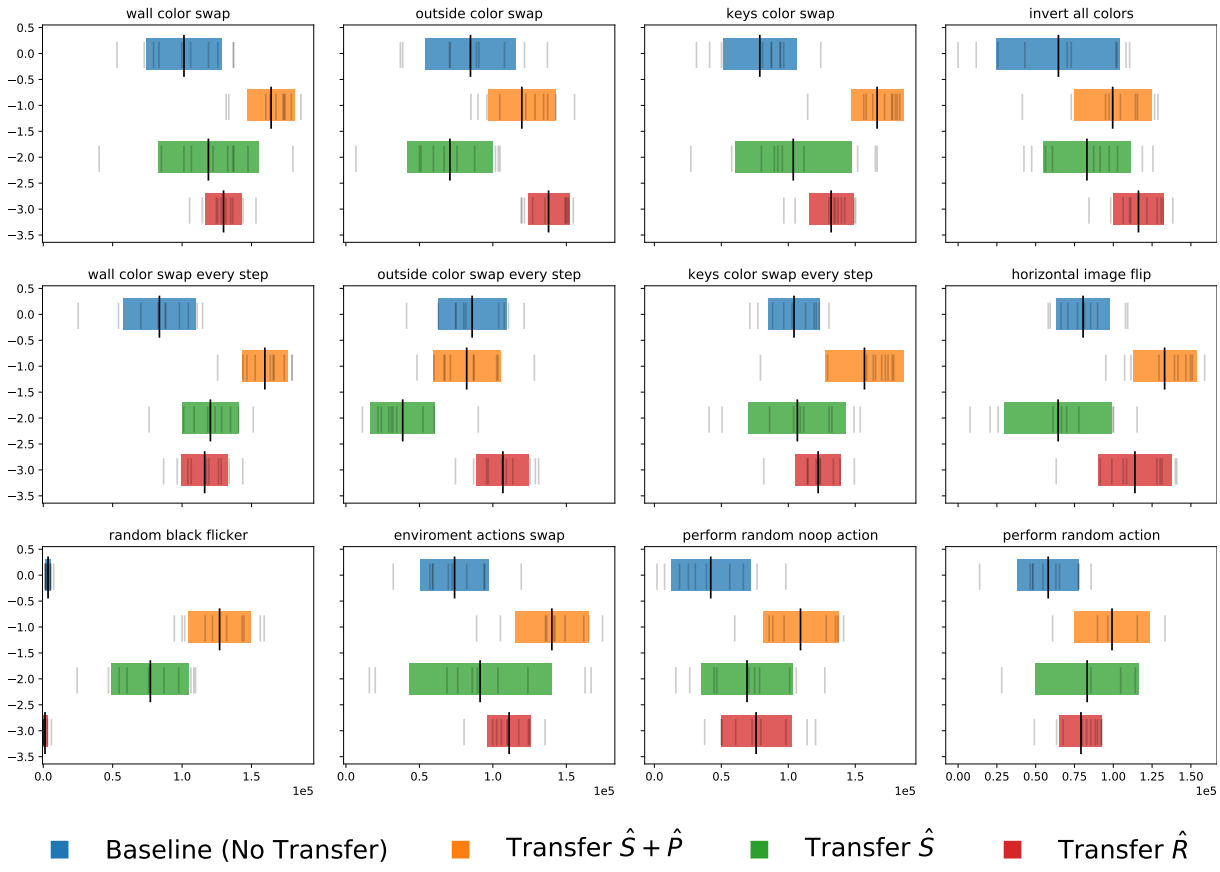


Figure A.2: Transfer performance of selected components of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results are computed with the intervals area under the curves, averaged over 10 randomly-seeded runs.

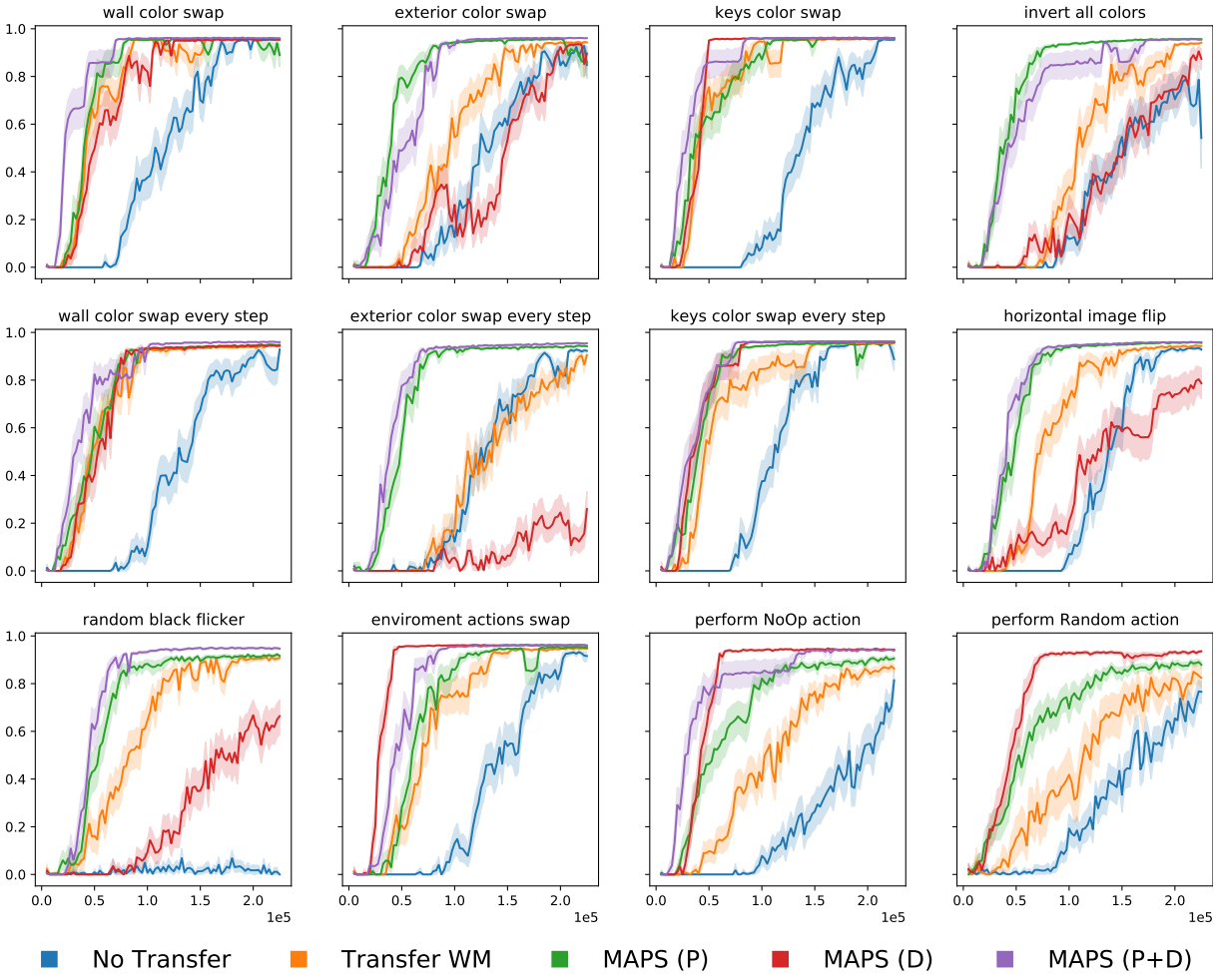


Figure A.3: Transfer performance of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

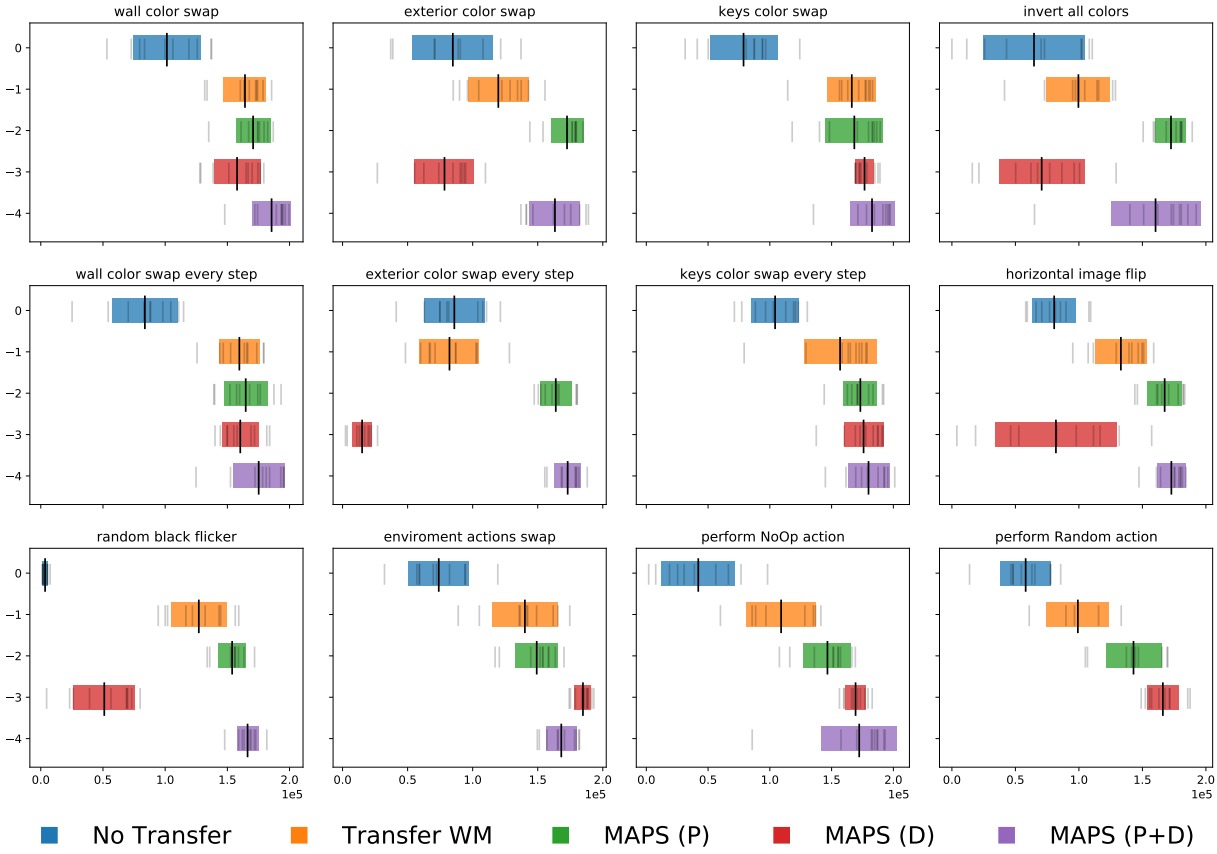


Figure A.4: Transfer interval of pretrained agents in MicroGrid to an augmentation task ($T_p \neq T_d$). Results are computed with the intervals area under the curves, averaged over 10 randomly-seeded runs.

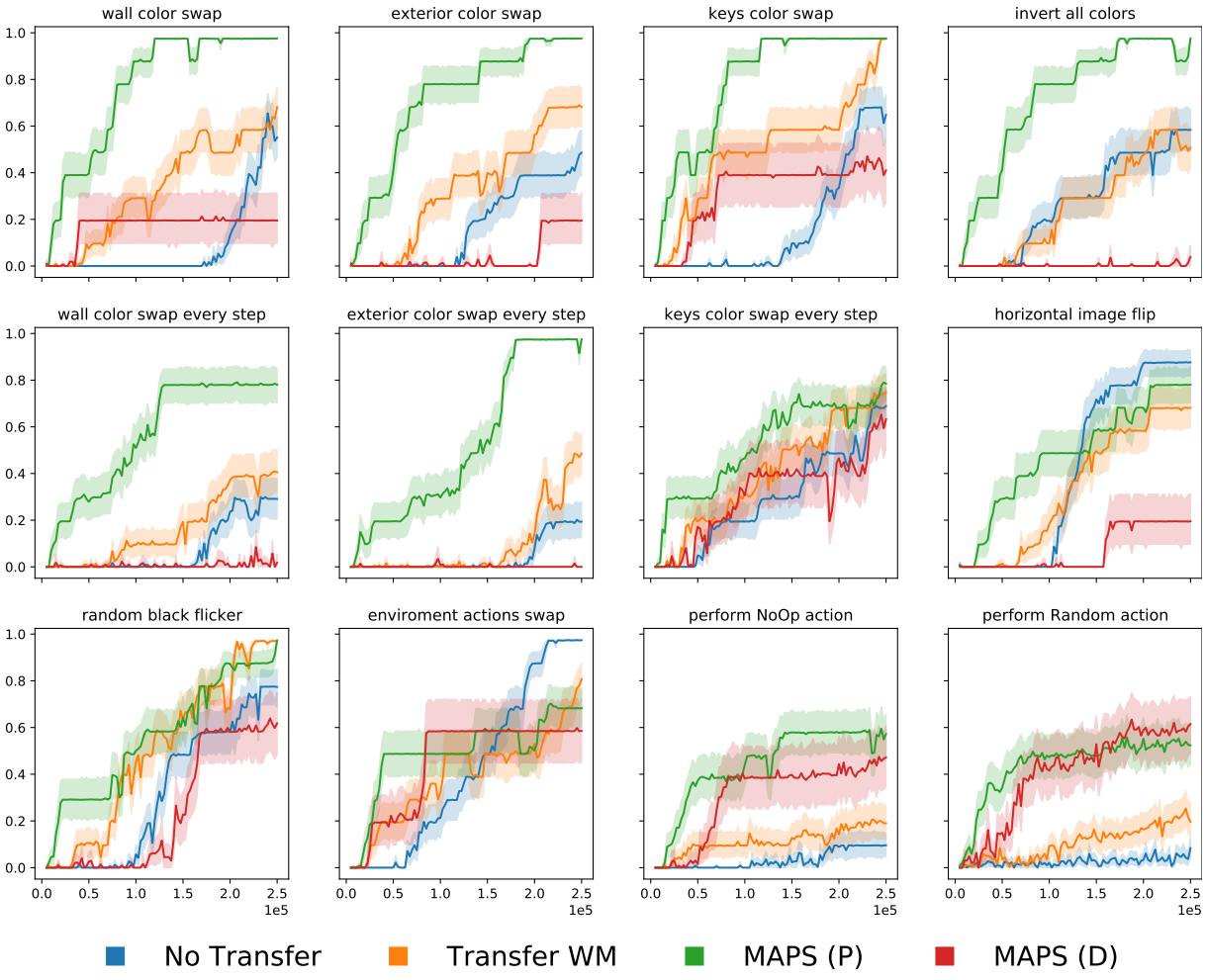


Figure A.5: Transfer performance of pretrained agents in MacroGrid to an augmentation task ($T_p \neq T_d$). Results averaged over 10 randomly-seeded runs.

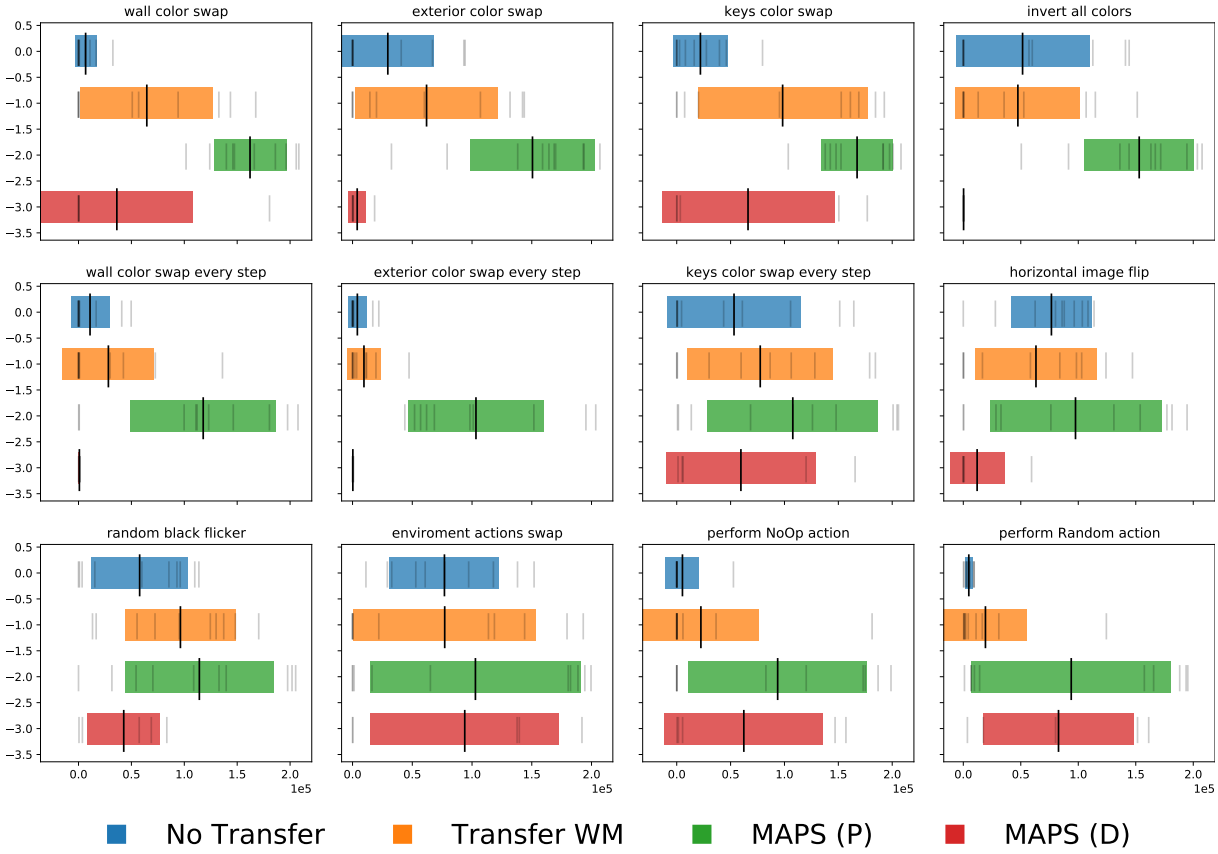
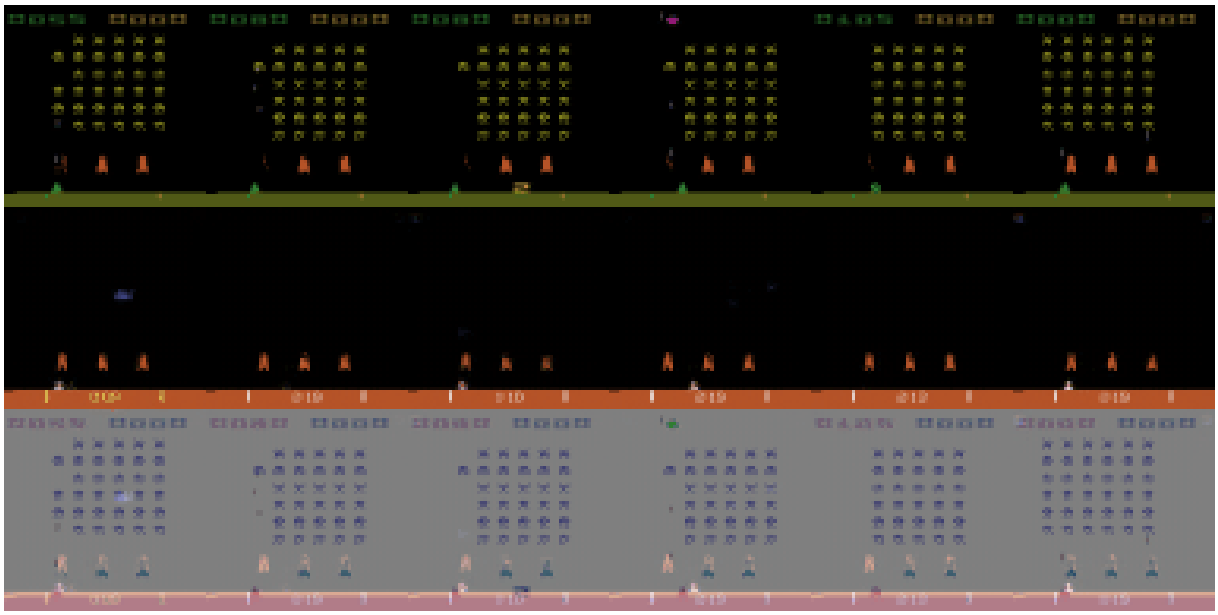


Figure A.6: Transfer interval of pretrained agents in MacroGrid to an augmentation task ($T_p \neq T_d$). Results are computed with the intervals area under the curves, averaged over 10 randomly-seeded runs.



(a) Single Task Transfer

(b) Single Task Transfer - MAPS (P+D)

Figure A.7: Zero-shot reconstruction of the world model, from Single Task Transfer without A.7a and with MAPS PD A.7b. The top row is the original game input, the middle row is the world model reconstruction of the game input, while the last row is the visual difference between both inputs.

(a) Multiple Task Transfer

(b) Multiple Task Transfer - MAPS (P+D)

Figure A.8: Zero-shot reconstruction of the world model, from Multiple Task Transfer without A.8a and with MAPS PD A.8b.

B

MAPS Augmentations Configurations

B.1 MiniGrid

Following the augmentations available in Fig. B.1 and Fig. B.2 , we categorize them as follows:

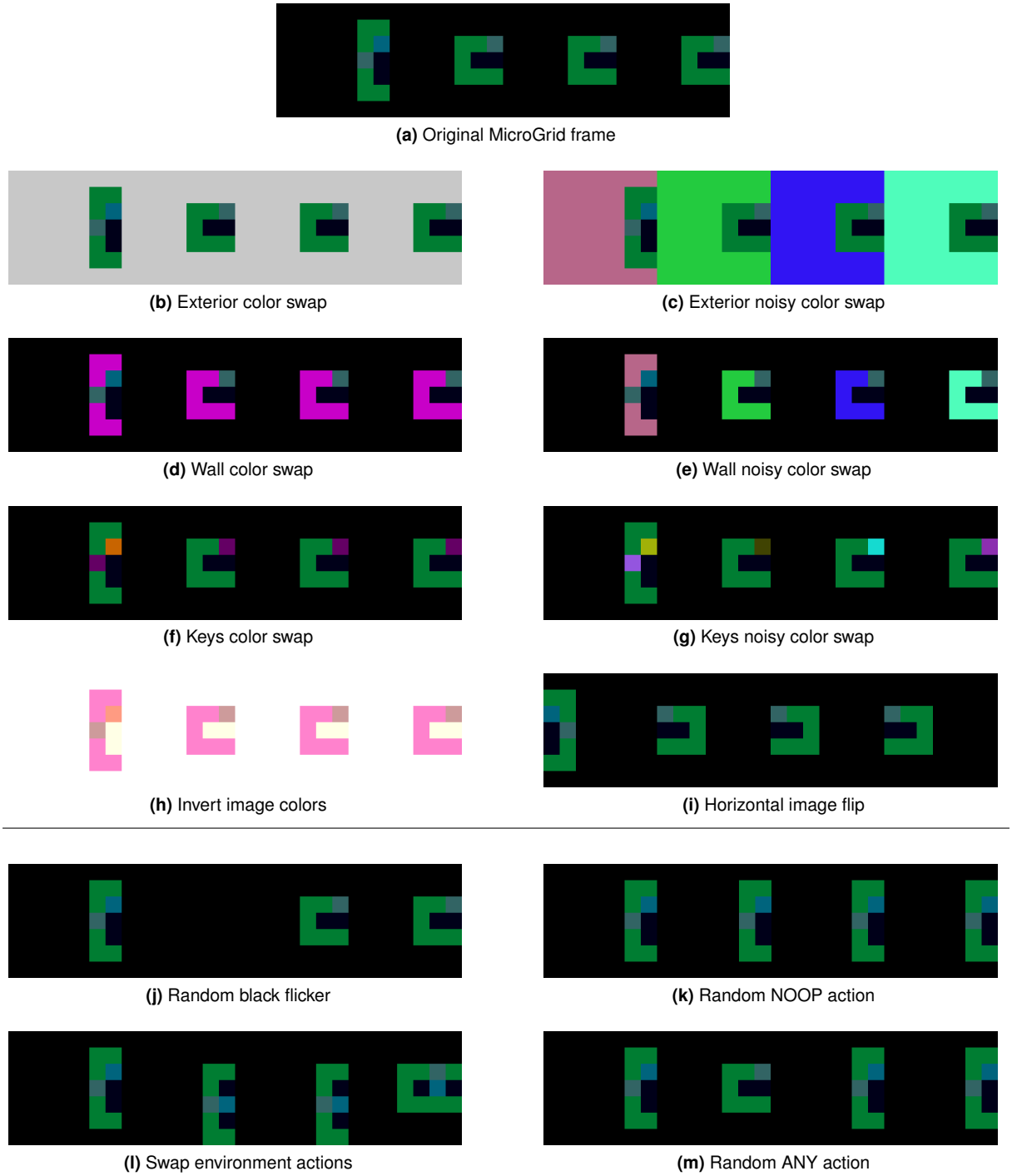
- Perceptual (P):
  - Static color changes: Figs. figs. B.1b, B.1d, B.1f and B.1h
  - Color changes on every step: Figs. figs. B.1c, B.1e and B.1g
  - Spatial visual changes: fig. B.1i
- Dynamical (D):
  - Modifications that do not change optimal policy: fig. B.1k
  - Modifications that alter optimal policy: figs. B.1l and B.1m
- Semantic (S):
  - Image occlusions: fig. B.1j

  - Swap object sprites positions (only in MacroGrid): fig. B.2b
  - Use different object sprites (only in MacroGrid): fig. B.2c

B.2 Atari

For the Atari experiments we use the following augmentations available in Fig. B.3:

- Perceptual (P): figs. B.3b to B.3e
- Dynamical (D): fig. B.3f
- Semantic (S): fig. B.3g



(a) Original MicroGrid frame

(b) Exterior color swap

(c) Exterior noisy color swap

(d) Wall color swap

(e) Wall noisy color swap

(f) Keys color swap

(g) Keys noisy color swap

(h) Invert image colors

(i) Horizontal image flip

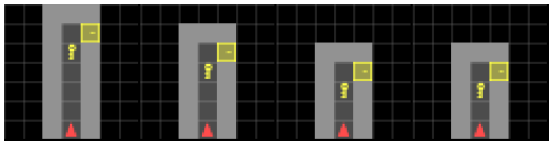
(j) Random black flicker

(k) Random NOOP action

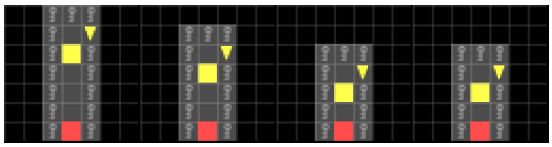
(l) Swap environment actions

(m) Random ANY action

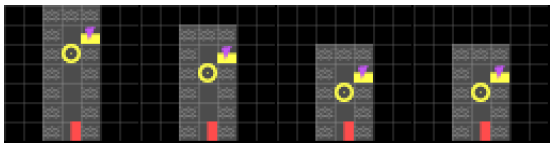
Figure B.1: Created augmentations for the MicroGrid environment: b-i) perceptual augmentations; k-m) dynamic augmentations; j) semantic augmentation.



(a) Original MacroGrid with semantics frame

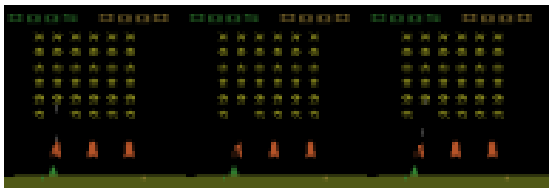


(b) Swap semantic shapes

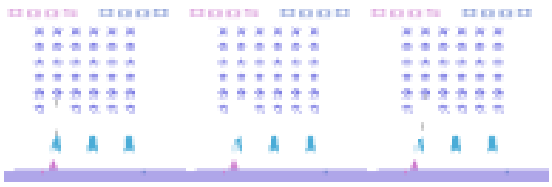


(c) Different semantic shapes

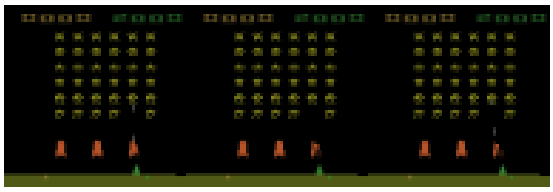
Figure B.2: Created augmentations for the MacroGrid environment: o-p) semantic augmentations.



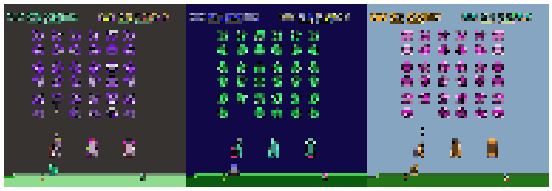
(a) Original Atari frames



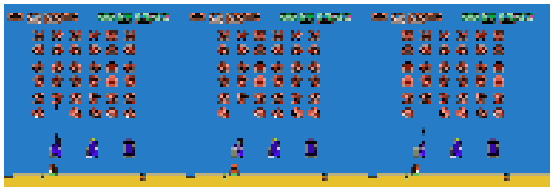
(b) Inverted color



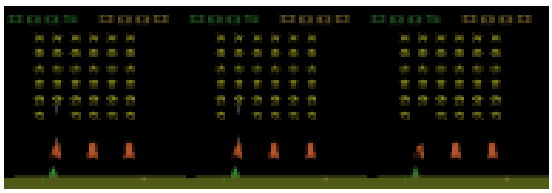
(c) Horizontal Flip



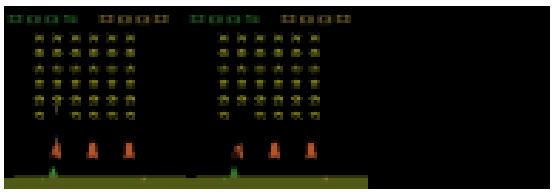
(d) Fast color jitter



(e) Fast color jitter (equal on sequence)



(f) Random repeat frames



(g) Random black flicker

Figure B.3: Created augmentations for the Atari environment, used in MAPS: b-e) perceptual augmentations; f) dynamic augmentation; g) semantic augmentation.

C

Implementation Details

C.1 Code Replication

In this work, we re-implement the original DreamerV2 [19] Tensorflow implementation in Pytorch. To make sure, that our replication works as close as possible to the original implementation, we made multiple unit and integration tests between the two frameworks. Not only that but we also used the same code structure from the original implementation, to ensure we use the same algorithmic choices. The success of the replication is verified in Fig. C.1.

C.2 Hyperparameters

In this section we describe in Table C.1 the base hyperameters used in this work. For the Atari task, we use the same hyperparameters as in the original DreamerV2 [19] paper, with the exception of the replay buffer size that was reduced from $2e6$ to $2e5$, due to unavailable computational resources. For the MiniGrid environment [27], we describe the changed the hyperparameters from the Atari task in Table C.2.

Table C.1: Hyperparameters used in DreamerV2

Name	Value
replay size	2e5
batch size	50
batch sequence size	50
kl balance	0.8
train agent every	16 steps
image input size	[64, 64]
clip rewards	tanh
prefill	$5e4$ steps
discount	0.999
weight initialization	Glorot Uniform [81]
Optimizer details	
optimizer	Adam [82]
optimizer epsilon	$1e-5$
weight decay	$1e-2$
gradient clip	100
model lr	$2e-4$
actor lr	$4e-5$
critic lr	$1e-4$
actor entropy	$1e-3$
kl loss scale	0.1
discount loss scales	5.0
reward loss scale	1
image loss scale	1
Architecture details	
RSSM hidden layers	600
RSSM deter	600
RSSM stoch	32
RSSM discrete	32
encoder layer type	Convolution
encoder depth	48
encoder activation function	elu
encoder kernels	[4, 4, 4, 4]
decoder layer type	Convolution
decoder depth	48
decoder activation function	elu
decoder kernels	[5, 5, 6, 6]
other heads layers	4
other heads units	400
other heads activation function	elu

Table C.2: Hypeparameters changes from Table C.1 used for MiniGrid environments

Name	Value
prefill	$5e3$
train every	10 steps
discount	0.99
kl loss scale	1
actor entropy	$3e-3$
MicroGrid only differences	
encoder layer type	Fully Connected
encoder layers	1
encoder units	50
decoder layer type	Fully Connected
decoder layers	1
decoder units	100
other heads layers	1
other heads units	100

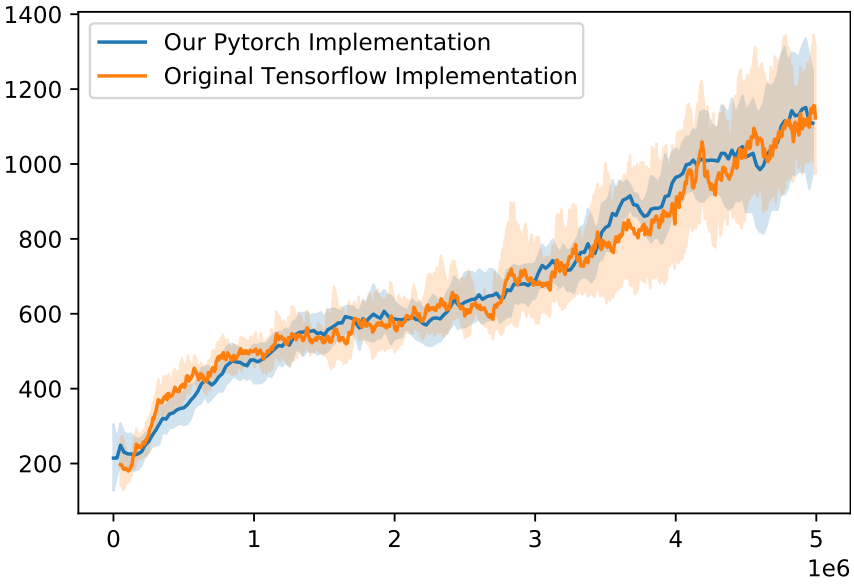


Figure C.1: Comparison of the Space Invaders Atari game learning performance over our implementation (Pytorch) vs the original implementation (Tensorflow). Results averaged over 5 randomly-seeded runs.

For more algorithmic details and parameters, the code is available at https://github.com/esteveste/dreamerV2-pytorch

C.3 Model Architecture

For better comprehension of the model architecture, we also provide figures describe the parameters and layers of the original Dreamer agent. Following the Fig. 2.6, here we present the details of the encoder (Fig. C.2), the posterior (Fig. C.4), the recurrent model (Fig. C.5), the prior (Fig. C.6), the decoder (Fig. C.3) and, finally, the details of the reward, discount, actor and critic heads (Fig. C.7). In this work, we use this model architecture and hyperparameters for the MacroGrid and Atari Tasks. For the MicroGrid, we use a smaller model with less parameters described in Table C.1, where the major architecture differences are described in Fig. C.8.

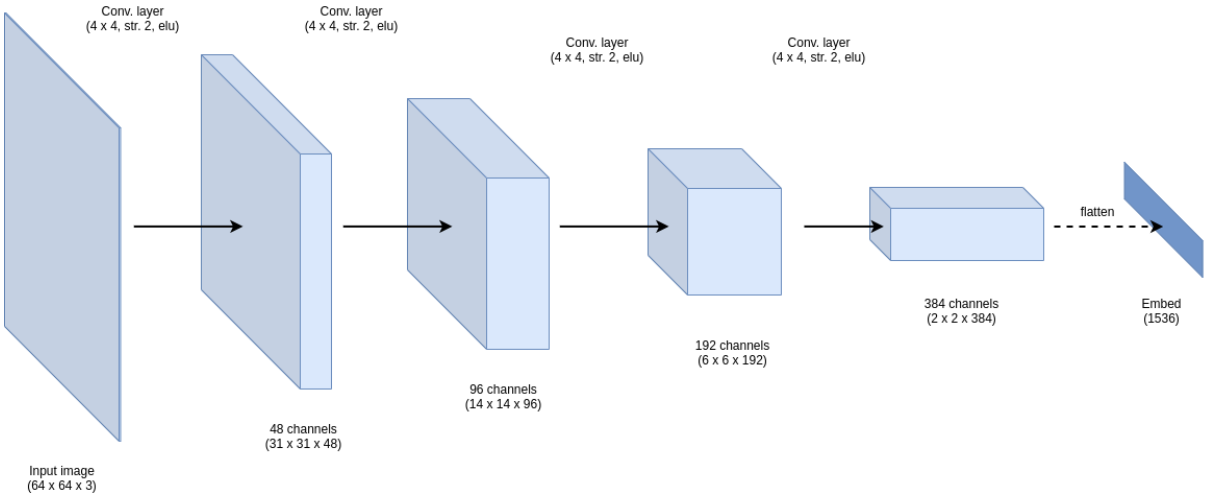


Figure C.2: Dreamer encoder architecture

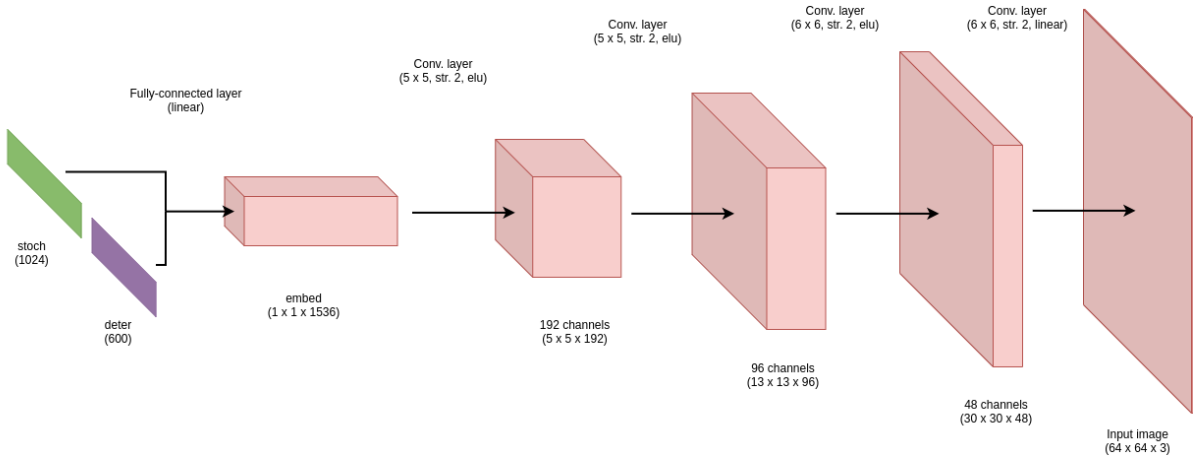


Figure C.3: Dreamer decoder architecture

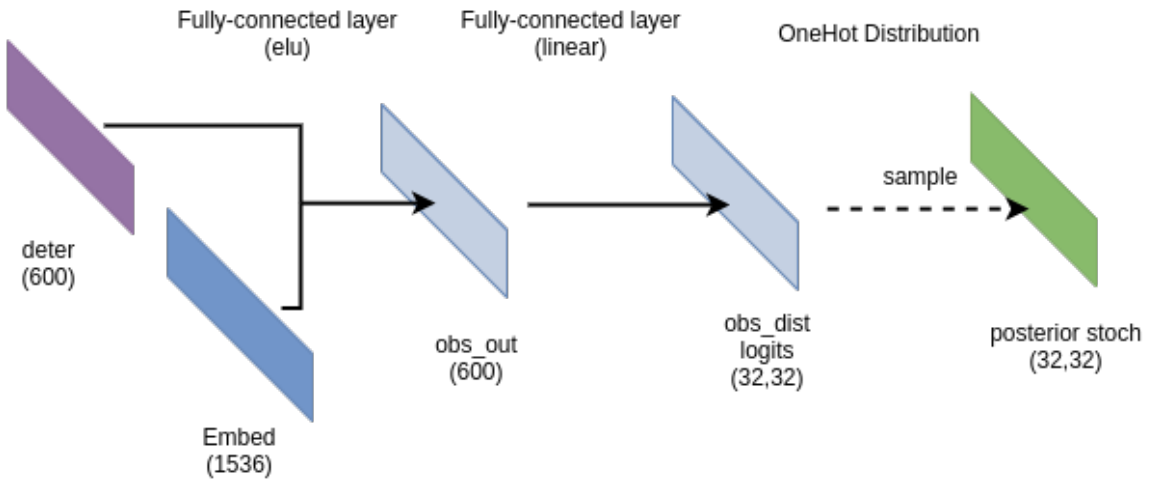


Figure C.4: Dreamer posterior architecture

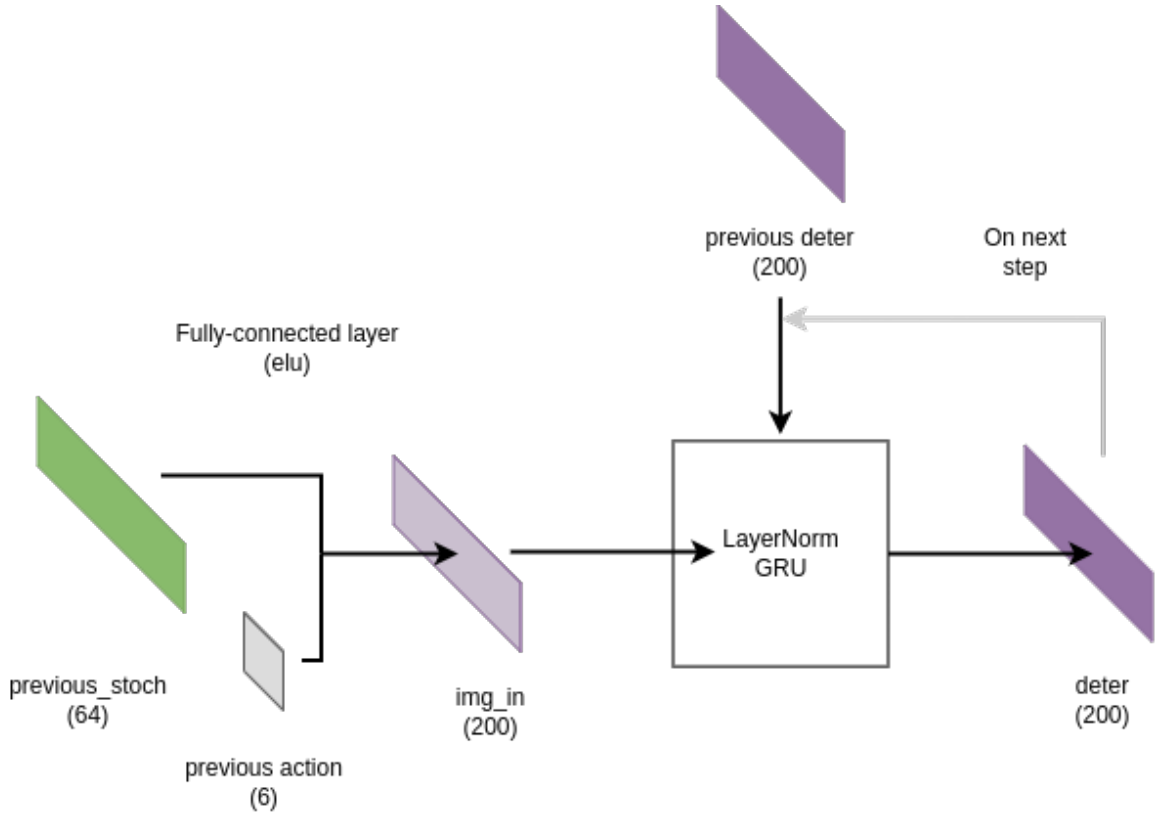


Figure C.5: Dreamer recurrent model architecture

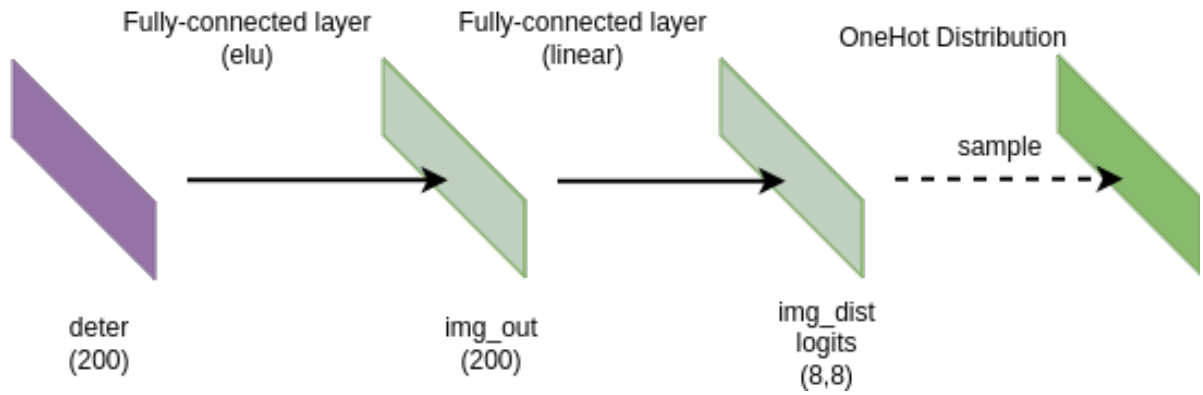


Figure C.6: Dreamer prior architecture

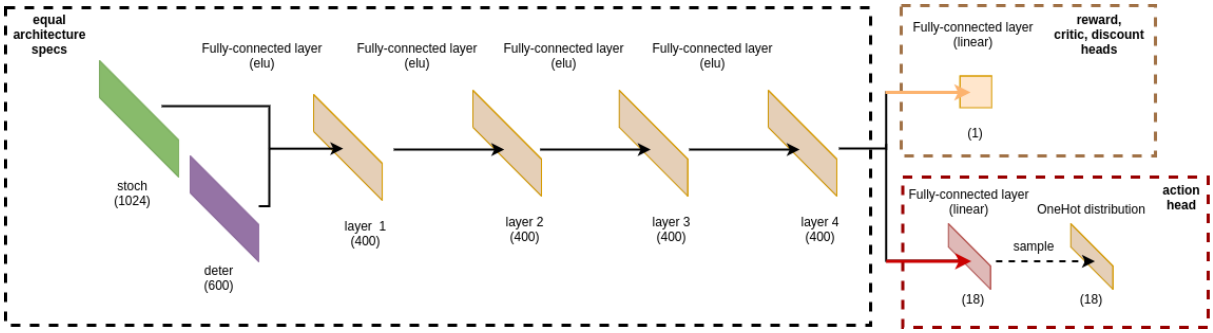
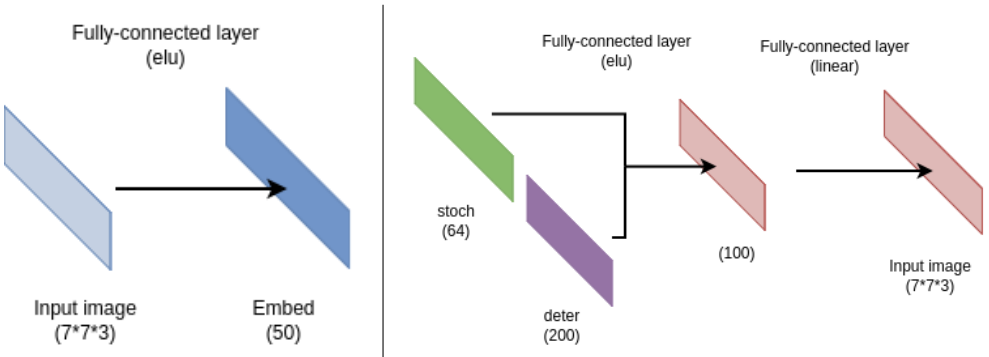
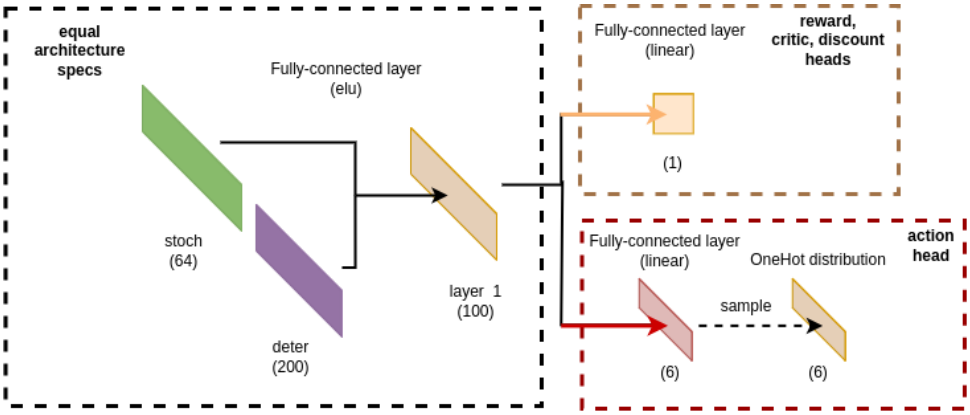


Figure C.7: DreamerV2 network architecture descriptions used for the reward, discount, actor and critic heads



(a) Small encoder **(b)** Small decoder



(c) Small reward, discount, acto and critic heads

Figure C.8: DreamerV2 small network architecture differences, used for MicroGrid tasks