HYPERNETWORK-PPO FOR CONTINUAL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continually learning new capabilities in different environments, and being able to solve multiple complex tasks is of great importance for many robotics applications. Modern reinforcement learning algorithms such as Proximal Policy Optimization can successfully handle surprisingly difficult tasks, but are generally not suited for multi-task or continual learning. Hypernetworks are a promising approach for avoiding catastrophic forgetting, and have previously been used successfully for continual model-learning in model-based RL. We propose `HN-PPO`, a continual model-free RL method employing a hypernetwork to learn multiple policies in a continual manner using PPO. We demonstrate our method on DoorGym, and show that it is suitable for solving tasks involving complex dynamics such as door opening, while effectively protecting against catastrophic forgetting.

1 INTRODUCTION

Adapting to changing environments is a crucial and desirable ability for many robot systems. Advances in reinforcement learning (RL) such as Proximal Policy Optimization (PPO) (Schulman et al., 2017) or Soft Actor-Critic (SAC) (Haarnoja et al., 2018) have enabled simulated robots to excel at solving complex continuous control tasks such as moving a stick figure and humanoid walking (Schulman et al., 2017). However, the resulting agents generally can only solve one narrowly defined task. When a new task is learned by fine-tuning the existing model , the ability to solve the old task is often greatly diminished or completely lost - a phenomenon known as *catastrophic forgetting*. Although a well-known problem (McCloskey & Cohen, 1989), catastrophic forgetting in sequential multi-task settings is still a major limitation for machine learning systems. This is in stark contrast to human learning, which very rarely exhibits forgetting when learning a new task (Parisi et al., 2019).

A variety of *continual learning* (CL) methods have been proposed to overcome this limitation. In a CL setting, multiple tasks are to be learned sequentially, while retaining or even improving knowledge about past tasks (Khetarpal et al., 2020). A naive approach towards this goal would be recording all previous experience of an agent, and replaying the recorded experience to a new agent when training new tasks. However, this method requires vast amounts of memory to store experiences, and thus does not scale well to a large number of tasks. Therefore CL desiderata often assume that previous experience will not be available when training new tasks (Li & Hoiem, 2016).

Hypernetworks have been previously proposed as a promising method (von Oswald et al., 2020) for supervised continual learning. More recently, task-conditioned hypernetworks have also been employed in a model-based continual RL setting using the cross-entropy method (CEM) (Huang et al., 2021).

In this paper, we expand upon the work of Huang et al. (2021); evaluating the use of task-conditioned hypernetworks in *model-free* continual RL. To this end, we propose a hypernetwork-based implementation of PPO (`HN-PPO`), a state-of-the-art online RL algorithm . We show that `HN-PPO` performs comparably to baseline PPO for learning single tasks, but unlike baseline PPO, it is also able to overcome catastrophic forgetting while learning a sequence of tasks. Further, we conduct an ablation study confirming the importance of regularizing the hypernetwork on previous tasks in order to avoid catastrophic forgetting.

Similar to Huang et al. (2021), we use the DoorGym environment (Urakami et al., 2019) in our experiments to demonstrate our method. This environment was chosen for its relevance to real-world robotics, where door opening is a highly relevant skill for autonomous mobile robots. Door opening is also a well-suited scenario for continual RL, since a robot may encounter many different kinds of doors during its life. The clear separation between different kinds of doors also fits well with the *task-incremental* CL scenario (van de Ven & Tolias, 2019), where the task identity is clearly defined, and known at training and evaluation time. Our main contributions are:

- A model-free continual RL algorithm, `HN-PPO`, using hypernetworks to learn a multi-task policy
- An evaluation of CL performance in DoorGym using the door opening success rate as the main performance indicator.

2 Related Work

CL strategies The many strategies for retaining previously learned knowledge, and thus alleviating the catastrophic forgetting problem, can be broadly categorized into four approaches according to van de Ven et al. (2022): Context-specific components, regularization, replay, and template methods. The latter is however most relevant for continual classification problems, and less so for RL settings. A similar classification has also been made in a review of CL methods by Khetarpal et al. (2020).

Context-specificity methods add parts to the network that are specific for each task, i.e. are only trained on a single task. Such methods include context-specific gating (Masse et al., 2018), which creates context-specificity by randomly excluding a subset of neurons from the network for each task, or progressive neural networks (Rusu et al., 2016), in which a completely new set of parameters is trained for each task, and information is transferred from previous tasks via lateral connections between layers.

Regularization can be applied at two levels to avoid catastrophic forgetting: In *parameter* regularization, the objective is to minimize the changes of parameters which are important for previously learned tasks. The main challenge of parameter regularization is finding which parameters are most important to retain previous knowledge: Kirkpatrick et al. (2017) use the Fisher information matrix to estimate parameter importance, and Zenke et al. (2017) propose an online-learned importance measure in their *Synaptic Intelligence* algorithm. Alternatively, regularization can be applied at the *functional* level. Herein, the output of the network is encouraged to stay close to a target for a set of anchor points. Anchor points could be chosen from previous experience, or if it is undesired to store this data, a knowledge distillation loss can be used for functional regularization, as in Li & Hoiem (2016). They propose a hybrid of functional regularization and parameter storage: Firstly, the network has a context-specific output layer, and regularization is applied between the new and previous outputs of the context-specific components of previous tasks, but using input data from the current task.

Finally, replay methods store experience from previous tasks in a buffer, and agents consequently revisit the recorded experience when training for new tasks. The policy is then jointly optimized on the current and replayed experience. Even though conceptually simple, replay suffers from bad scalability over many tasks, since large amounts of memory are required to store all past experience. An approach to improve scalability is to train a generative model alongside the policy model, and then sample from this model to obtain pseudo-data to revisit (Riemer et al., 2019).

DoorGym The DoorGym environment (Urakami et al., 2019) is a robot simulation based on OpenAI Gym (Brockman et al., 2016), where robot arms aim to open a door. Robots are controlled via a set of continuous input variables modeling joint torque (or torque and force on the end effector in case of `floating` robots). Different types of tasks are modeled through the door knob style (`pull`, `lever` or `round`) as well as the location of the hinge (`right` or `left`) and the opening direction (`pull` or `push`). Environments within a task domain have a multitude of randomized features, such as the location of the handle on the door, the exact shape of the handle, or the spring force on the lever and hinge. The original authors of the DoorGym environment (Urakami et al., 2019) provide a baseline PPO and SAC agent for the `pull` and `lever` doors. It is demonstrated that PPO and SAC

are in principle able to solve the door opening tasks, both in their ground configuration, and with added noise. We use their set of hyperparameters for the PPO agent for our experiments using PPO, as well as as a starting point for finding hyperparameters for HN-PPO. Of note, the authors did not make any attempt at a continual RL agent, nor was the DoorGym environment designed specifically for continual learning.

Huang et al. (2021) propose *HyperCRL*, a model-based continual RL algorithm based on the cross-entropy method (CEM) (Rubinstein, 1997) and hypernetworks (von Oswald et al., 2020), and evaluate it on the DoorGym environment. Herein, it is shown that catastrophic forgetting occurs between different DoorGym tasks with traditional RL methods, and task-conditioned hypernetworks are established as a promising method to alleviate this. However, their paper does not show if the tasks can be completely solved, as they only report normalized episode rewards, not success rates.

3 BACKGROUND

Our proposed method uses PPO (Schulman et al., 2017) to continually learn policies for different door-opening tasks of the DoorGym environment (Urakami et al., 2019). A hypernetwork (von Oswald et al., 2020) architecture is used to prevent catastrophic forgetting.

Hypernetworks In a task-incremental CL setting $\mathcal{X} \times \mathcal{C} \rightarrow \mathcal{Y}$ (van de Ven & Tolias, 2019), a *task-conditioned hypernetwork* is a neural network h that maps the task identity $\mathcal{C}$ to a set of parameters of a target network, i.e. $h : \mathcal{C} \rightarrow t$. The target network $t : \mathcal{X} \rightarrow \mathcal{Y}$ then maps observations $\mathcal{X}$ of the RL agent to its actions $\mathcal{Y}$. The task identity $\mathcal{C}$ is represented by a low-dimensional, trainable embedding vector in our implementation, which is based on work by Auddy et al. (2022).

To preserve previously learned skills, regularization is applied at the meta-level: At the beginning of training a new task, the outputs Θ_t of the hypernetwork are recorded for each previous task embedding t (von Oswald et al., 2020). Since this output completely defines the target network for a each task, keeping the Θ_t constant during training of a new task will guarantee preservation of the learned dynamics. We hence apply an L2 penalty on changes of the hypernetwork's output:

$$\Theta_t = h(t, \Theta_h) \tag{1}$$

$$\mathcal{L}_{reg} = \beta \frac{1}{T-1} \sum_{t=0}^{T} ||\Theta_t - \Theta_{t,new}||_2 \tag{2}$$

The regularization coefficient β is a supplied hyperparameter. Of note, $\Theta_{t,new}$ is calculated after a *candidate change* to the hypernetwork weights Θ_h is already applied. This results in a 2-stage weight update routine; the candidate change is first calculated from the task loss $\mathcal{L}_{task}$, then the "true" change $\Delta\Theta_h$ is calculated using both the task and regularization loss (von Oswald et al., 2020).

Because the update routine requires passing all previous task embeddings through the hypernetwork to calculate the $\Theta_{t,new}$, the computational expense of a training step increases $\mathcal{O}(n)$ with respect to the number of tasks. Importantly though, the model size stays almost constant: only a low-dimensional task embedding vector is added for each new task. This fulfills the CL desideratum of not explicitly accumulating previous experience.

Proximal Policy Optimization PPO is a policy gradient method using a *clipped surrogate objective*. A policy is optimized by estimating its gradient, and conducting gradient ascent steps to maximize the reward. The gradient is estimated by differentiating the objective function

$$L_t(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \tag{3}$$

with advantage $\hat{A}_t$, and a probabilistic policy $\pi_\theta(a_t|s_t)$. In PPO, this objective is constrained by clipping excessively large changes in probability ratio (Schulman et al., 2017):

$$L_t^{clip}(\theta) = \mathbb{E}_t \left[min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1+\epsilon, 1-\epsilon \right) \hat{A}_t \right) \right] \tag{4}$$

ϵ is a hyperparameter and defines the allowable change of the policy. This improves the policy stability by removing big changes in the probability ratios of the old and new policies during a training step. It is conceptually similar to Trust-Region Policy Optimization (Schulman et al., 2015), but is easier to implement, since it omits the KL-divergence based constraint used in Trust-Region Policy Optimization .

Additionally to the action objective in Eq. 4, we use a trainable value estimator (called a "critic network") as proposed by Schulman et al. (2016) to introduce a value loss term L^{vf}. An entropy bonus S is also added to incentivize exploration of the parameter space. The complete objective function thus has 3 parts, with value coefficient c_v and entropy coefficient c_e being hyperparameters.

$$L_t^{total}(\theta) = L_t^{clip}(\theta) + c_v L_t^{vf}(\theta) + c_e S_{\pi_\theta}(s_t) \tag{5}$$

4 METHOD

We extend the existing implementation of PPO in DoorGym to use a hypernetwork to generate the weights of the policy network. Two variants of `HN-PPO` were implemented: A straightforward implementation (`HN-PPO`) which places both the actor and critic networks' parameters under the control of the hypernetwork, and a modified variant that only generates the actor network via a hypernetwork and uses an ordinary MLP as the critic (`HN-PPO+fc`). In the latter variant, the critic is re-initialized for each new task seen by the agent (cf. Appendix A.1 for details). Since the critic is only used during training, it does not influence the policy at evaluation time, nor is it strictly required to remember multiple dynamics. However, similarity between tasks could allow for forward transfer and benefit learning of new tasks in case of the `HN-PPO` as the critic is "pre-trained". The PyTorch hypernetwork implementation used in our experiments is from Auddy et al. (2022) and was used without modifications.

4.1 ENVIRONMENT

In our experiments, we used an unmodified DoorGym (Urakami et al., 2019) environment with the `blue-floatinghook` robot. This robot type was chosen due to both its simplicity and the good baseline performance on `pull` and `lever` doors. The robot resembles a hand with a hook for fingers, and is freely floating in 3D space. It has a 6D continuous input space, with three inputs controlling force in x/y/z direction, and three inputs controlling yaw/pitch/roll torque. None of the noise or vision features of DoorGym were used in our setup, i.e. observations are noise-free and directly derived from the environment simulation. We use the same 6-task sequence laid out in Table 1 for all continual learning experiments. CL agents were sequentially trained on each environment. Figure 1 provides example images of the robot interacting with some of the door environments.

4.2 METRICS

To evaluate the capability of the trained agents, we use the evaluation success rate as the primary metric. In each evaluation, agents are tested on 100 pseudo-randomly picked worlds (of 3000 total), and an opening attempt is deemed successful if the door was opened at least 0.2 rad within 20

Table 1: Series of door opening tasks for CL.

Task ID	Handle	Hinge location	Opening direction
0	`pull`	right	pull
1	`pull`	left	pull
2	`lever`	right	pull
3	`lever`	left	pull
4	`lever`	right	push
5	`lever`	left	push

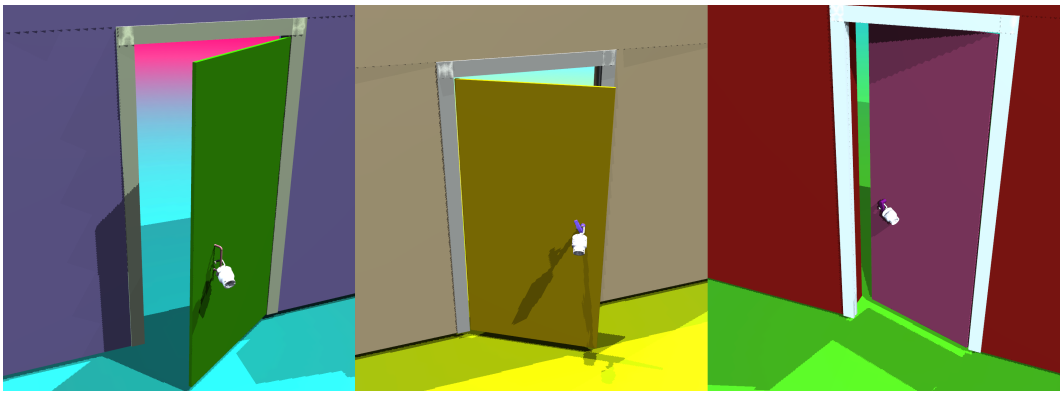


Figure 1: Examples of the DoorGym environment and robot. Left: `pull` handle (task 0); middle: `lever` handle with left hinge (task 3); right: `lever` handle with inward (`push`) opening direction (task 4).

seconds of simulation time (Urakami et al., 2019). Multiple continual learning metrics are derived from the success rates of an agent in different stages of its training.

Continual Learning Metrics Díaz-Rodríguez et al. (2018) propose a set of CL metrics based on a train-test accuracy matrix: After training an agent on N tasks, the entries $R_{i,j}$ of the $\boldsymbol{R} \in \mathbb{R}^{N \times N}$ accuracy matrix is defined as accuracy on task j after training on task i. If $j > i$ and thus the task j to be evaluated is not known to the hypernetwork, then i is used in the hypernetwork to generate the target network weights (i.e. the embedding of the last known task is used). Using the accuracy matrix A, forward transfer (FT), positive backward transfer (BWT^+) and remembering (REM) are calculated. The latter 2 are derived from the *backward transfer* BWT, but are separated to keep all metrics within $[0, 1]$.

$$A = \frac{2 \sum_{i=0}^{n} (\sum_{j \leq i}^{n} R_{i,j})}{N(N+1)} \tag{6}$$

$$FT = \frac{2 \sum_{i=0}^{n} (\sum_{j=i+1}^{n} R_{i,j})}{N(N-1)} \tag{7}$$

$$BWT = \frac{2 \sum_{i=1}^{n} (\sum_{j=0}^{i-1} (R_{i,j} - R_{j,j}))}{N(N-1)} \tag{8}$$

$$BWT^+ = max(0, BWT) \tag{9}$$

$$REM = 1 - |min(BWT, 0)| \tag{10}$$

5 Experimental evaluation

We conducted multiple experiments to evaluate the ability of `HN-PPO` to learn the complex dynamics of different door opening tasks. A special focus was set on analyzing how well catastrophic forgetting can be avoided by using a task-conditioned hypernetwork, and whether forward transfer could be observed during the training sequence. All experiments were run three times with different PRNG seeds. Reported values are the mean $\pm$ standard error of the mean of 3 independent results.

5.1 Baselines

Table 2 shows the average after-training opening rates for each task and each method discussed in the following sections. Opening rates are reported at the end of training for a specific task, before any other tasks are visited by the agent. Comparing the results for the non-CL methods (PPO, PPO-finetuning, and `HN-PPO`+fresh network), the `pull` doors (task 0 and 1) are the easiest to open, while the `lever` doors with pull opening direction (task 2 and 3) are the hardest to open.

Table 2: After-training opening rates per task.

Task ID	PPO	PPO-finetuning	HN-PPO+fresh network	HN-PPO	HN-PPO+fc
0	1.00 ± 0.00	1.00 ± 0.00	0.98 ± 0.014	0.98 ± 0.014	1.00 ± 0.00
1	1.00 ± 0.0027	1.00 ± 0.00	0.97 ± 0.018	1.00 ± 0.00	1.00 ± 0.00
2	0.32 ± 0.26	0.00 ± 0.00	0.27 ± 0.22	0.92 ± 0.014	0.69 ± 0.21
3	0.32 ± 0.25	0.00 ± 0.00	0.25 ± 0.20	0.31 ± 0.25	0.36 ± 0.23
4	0.96 ± 0.014	0.37 ± 0.23	0.59 ± 0.24	0.59 ± 0.24	0.25 ± 0.21
5	0.63 ± 0.22	0.36 ± 0.24	0.69 ± 0.15	0.36 ± 0.15	0.01 ± 0.011
Average	0.71 ± 0.17	0.46 ± 0.14	0.63 ± 0.17	0.69 ± 0.15	0.55 ± 0.15

Table 3: Continual learning metrics for baseline experiments. The low accuracies and remembering scores indicate catastrophic forgetting.

Metric	PPO	PPO-finetuning	HN-PPO+fresh network
Accuracy	0.24 ± 0.035	0.20 ± 0.035	0.21 ± 0.023
Forward transfer	0.10 ± 0.0022	0.10 ± 0.028	0.03 ± 0.0021
Pos. backward transfer	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Remembering	0.29 ± 0.078	0.47 ± 0.060	0.32 ± 0.019

Tables 3 and 4 list the CL metrics (cf. Section 4.2) for all experiments discussed in the following. The tables are split for layout purposes only.

PPO The single-task PPO implementation in DoorGym, based on code from Kostrikov (2018), was used as a baseline for the performance of individual tasks. Hyperparameters for PPO are taken from Urakami et al. (2019). For this baseline, a new agent is created for each task, and only trained on that specific task. We observe that the baseline algorithm reliably learns to solve the tasks with `pull` handles, but yields unstable results for the more challenging `lever` handles, depending on the seed being used. In the case of tasks 2, 3 and 5, seed dependence was prominent enough so that the agent completely failed to solve the task for one seed, while it could achieve close to 100% opening rate using a different seed. Figure 2a shows the door opening rate of the agents at different points in the training progress. The large differences between seeds can clearly be observed, especially in tasks 2 and 3, which use the `lever` knob. The average after-training opening rate across all 6 tasks is 71% (2).

PPO-finetuning This baseline uses "standard" PPO as before, but instead of creating a new model for each task in the sequence and training it independently, a single agent is fine-tuned on each task in the sequence. The resulting model from training for one task is then used as a pre-trained model for the next task. Since PPO makes no considerations towards continual learning, fine-tuning of policies learned with PPO serves as a lower baseline for the ability of an agent to retain knowledge of previous tasks. The opening rates plotted in Figure 2b show the expected catastrophic forgetting at the task boundary for some tasks (e.g. between task 1 and 2), but also reveal interesting transfer

Table 4: Continual learning metrics for HN-PPO and HN-PPO+fc. Remembering is at its maximum possible value, 1.00, for both methods, indicating no forgetting is taking place.

Metric	HN-PPO	HN-PPO+fc
Accuracy	0.81 ± 0.041	0.73 ± 0.080
Forward transfer	0.052 ± 0.0095	0.04 ± 0.0054
Pos. backward transfer	0.0015 ± 0.0013	0.00 ± 0.00018
Remembering	$\mathbf{1.00} \pm 0.0024$	$\mathbf{1.00} \pm 0.0044$

behaviors. Between task 0 and 1 (both `pull` handles), only little forgetting of knowledge on task 0 occurs, while task 1 has a zero-shot success rate of 61% thanks to pre-training on task 0. This is likely due to the high similarity of dynamics for these two tasks. On the other hand, the agents' success rate on the `lever` tasks (2 and 3) was greatly reduced compared to the PPO baseline. Strikingly, even though no success was achieved in solving the `lever` task during training, one agent was able to open `lever` doors after being trained on the subsequent `lever_push` task (task 4) interval in Figure 2b.

These unexpected observations demonstrate the complex relationships between different door opening tasks, which can potentially be exploited by continual learning.

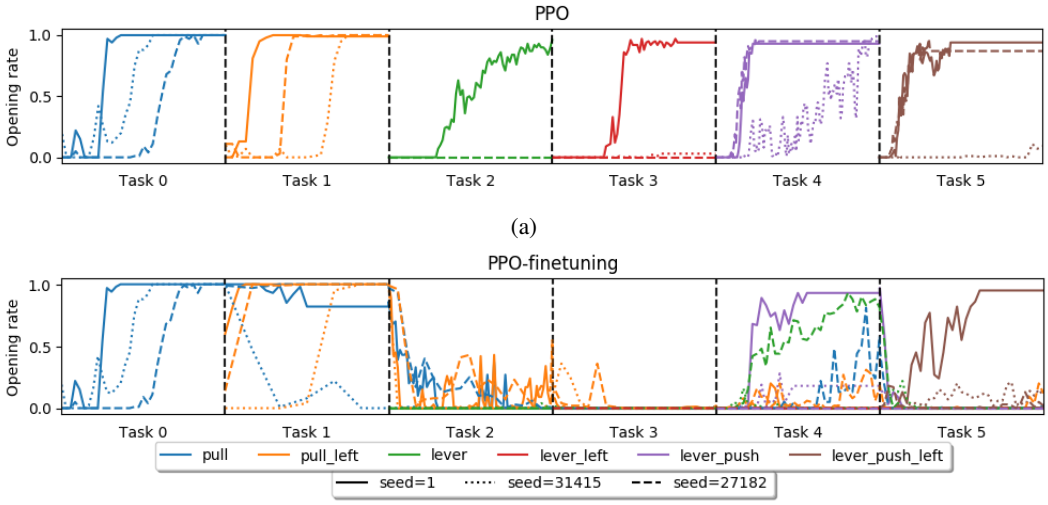
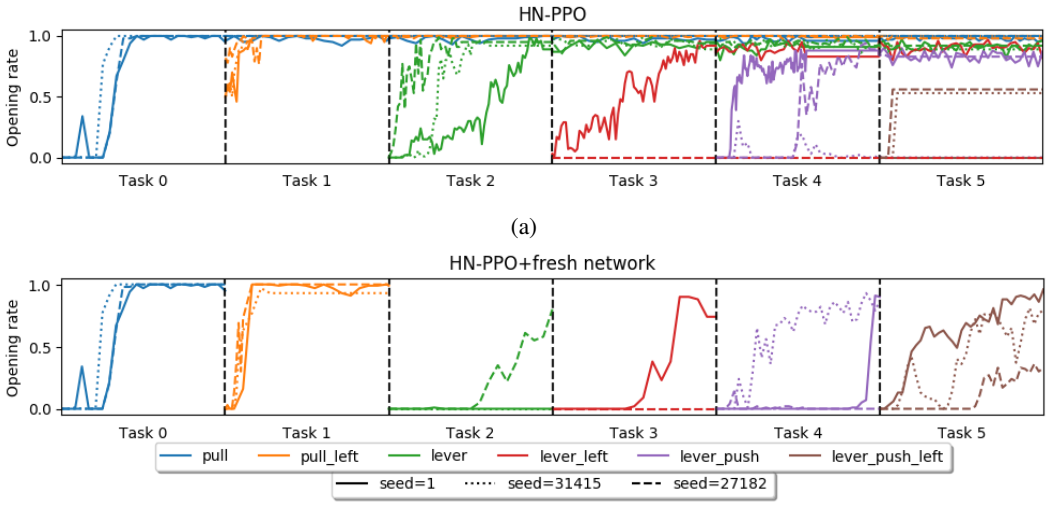


Figure 2: Door opening rates using **(a)** PPO and **(b)** PPO-finetuning for different door opening tasks and seeds. The x-axis represents the training progress normalized to the longest training run for each task. Progress can thus be compared for multiple training runs of the same task, but not between different tasks. Note that in (a), the sequence of tasks is chosen for consistency with other figures. Each task is trained and evaluated independently.

HN-PPO+fresh network To directly compare the impact of previous experience of a `HN-PPO` agent on its performance, we train a new `HN-PPO` agent for each task in this experiment. Like in the PPO baseline, each agent is only trained on a single task, and all agents are trained independently. Since this experiment uses the same network architecture that is used for continual learning, any difference to this baseline is a direct result of an agent's previous experience. Similar to the PPO baseline, large performance differences were again observed between different seeds, as can be seen in Figure 3b. The average after-training opening rate is slightly lower than what was achieved by PPO (63% vs. 71% for PPO), but within the errors of the respective values (Table 2).

5.2 Continual Learning Experiments

HN-PPO In this experiment, a single `HN-PPO` agent was trained sequentially on the tasks in the CL sequence. Figure 3a shows the opening rate for each of the 6 tasks in the CL sequence during the agents' training. The lack of sharp drops in the opening rate graph clearly indicates that the hypernetwork is able to learn multiple tasks with different dynamics without any significant loss of accuracy in previously seen tasks. `HN-PPO` shows excellent protection against catastrophic forgetting, achieving a remembering score of 1.00, as can be seen in Table 4. This indicates that on average, no skill was lost by learning additional skills. The method also achieves very similar after-training opening rates to the PPO baseline (Table 2), which confirms that the CL capabilities do not adversely affect PPO's single-task performance. While there is virtually no forgetting, significant backward transfer does also not occur. This result is expected, since the hypernetwork regularization aims to minimize changes to previously learned dynamics, both beneficial and detrimental.



(a)

(b)

Figure 3: **(a)** Door opening rates using `HN-PPO`, **(b)** `HN-PPO` with a fresh network for each task.

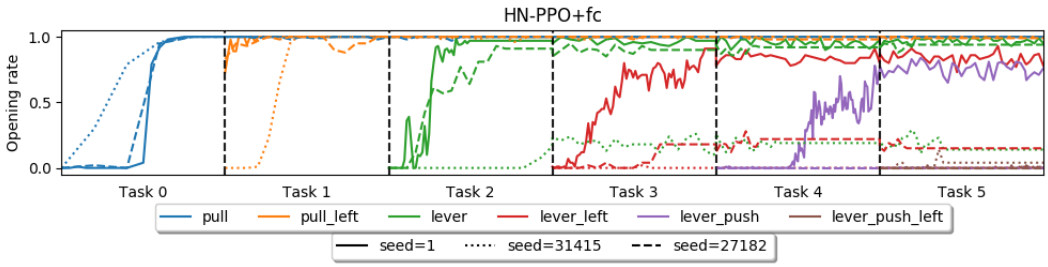


Figure 4: Door opening rates using HN-PPO+fc.

Fresh Critic In this experiment, a modified `HN-PPO` agent is used, which only parameterizes the actor network with a hypernetwork. The critic network on the other hand is freshly initialized for each task. This alternative architecture, `HN-PPO+fc`, is then trained on the CL task sequence in the same way as the `HN-PPO` agent. Compared to standard `HN-PPO`, the after-training opening rate is 20% lower (55%, vs. 69% for `HN-PPO`) and the CL accuracy is 10% lower (73%, vs. 81% for `HN-PPO`). Importantly though, we find that `HN-PPO+fc` exhibits the same, highly effective protection against catastrophic forgetting as `HN-PPO`, also having a remembering score of 1.00.

5.3 Regularizer Ablation Study

In this experiment, we aim to investigate the importance of regularizing the weights of previously learned tasks. Experiments were run with the same hyperparameters and environments as in Section 5.2, but regularization of the hypernetwork outputs on previous task embeddings was disabled by setting $\beta = 0$ (cf. Eq. 2). Removing hypernetwork regularization resulted in a 67% decrease in average accuracy, as well as a 76% decrease in remembering of previous tasks, as can be seen in Table 5.

Without regularization, the near-perfect remembering of previous tasks seen in `HN-PPO+fc` drops even below the PPO-finetuning baseline (cf. Table 3). This result confirms that while hypernetworks provide the model with the necessary capacity and flexibility, regularization of the hypernetwork outputs is mainly responsible for their high resilience against catastrophic forgetting.

Table 5: Effects of hypernetwork regularization on continual learning

Algorithm	HN−PPO+fc	HN−PPO+fc, no regularization
Accuracy	**0.73** $\pm$ 0.080	0.24 $\pm$ 0.038
Forward transfer	0.04 $\pm$ 0.0054	0.03 $\pm$ 0.012
Pos. backward transfer	0.00 $\pm$ 0.00018	0.00 $\pm$ 0.00
Remembering	**1.00** $\pm$ 0.0044	0.24 $\pm$ 0.076

6 Discussion

In our experiments, we have shown that our HN−PPO agent can successfully learn to open up to 6 different types of doors. While Huang et al. (2021) have demonstrated that hypernetwork-based CL is able to consistently achieve high returns on multiple DoorGym tasks, our work shows that this capability also extends to fully solving the task at hand, i.e. opening the door. To this end, we chose to use the door opening success rate (cf. Section 4.2) as the main performance indicator, in contrast to Huang et al. (2021), who report the average return as the core metric. We argue that in a scenario with a clear final goal, such as door opening, the success rate better represents the quality of a policy. The training time until a door can be reliably opened is however significantly longer than what Huang et al. (2021) experimented with. In their experiments, agents were trained for 60.000 environment steps per task, while we trained for up to $2.4 * 10^7$ environment steps per task.

The distinction between these 2 metrics is further motivated by our discovery that under certain circumstances, the evaluation return of the DoorGym environment is not representative of the ability of an agent to solve the task. Similar returns in the same environment can correspond to vastly different opening rates, as shown in Figure 5. While the three return curves follow a similar pattern, one agent is able to solve the task with a high opening rate, while the other 2 are unsuccessful.

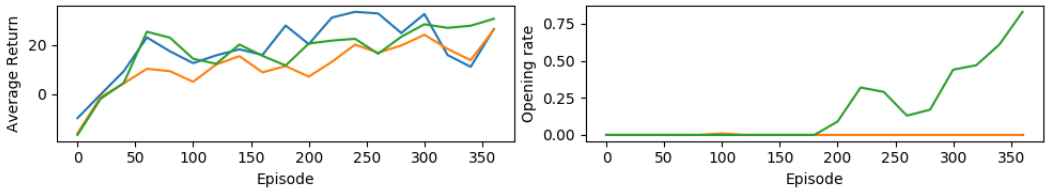


Figure 5: Return-opening rate mismatch: Evaluation return and opening rate curves of 3 experiments in the same environment (HN−PPO+fresh network, lever task).

Interestingly, the choice of the time point to end an agent's training on one task does sometimes greatly affect downstream tasks. Even when convergence is reached for key metrics (opening rate, average return), an agent might not be able to learn a following task when starting from one checkpoint, but succeed in doing so when starting from an earlier or later checkpoint. This hidden ability of a hypernetwork to learn further tasks may be of interest for further research. Avoiding states in which the learning of future tasks is hindered will help to further improve the capabilities of hypernetworks in CL.

7 Conclusions, Limitations and Outlook

We presented HN−PPO, a novel approach to continual RL using PPO and a hypernetwork-based policy network. Using our method, the agent learns a policy for solving multiple tasks via a task-conditioned hypernetwork. New tasks can be trained in a continuous setting. The hypernetwork size is almost constant: it only grows by the size of the task embedding vector, which is only 8-dimensional in our experiments. This enables our method to scale to high numbers of tasks. In our experiments, we demonstrated that HN−PPO is a highly effective method to learn multiple tasks with complex dynamics, while offering strong protection against catastrophic forgetting. We compared two network architectures; a shared-parameters actor-critic using a hypernetwork, and an

architecture with a non-hypernetwork critic. While we found the shared-parameter architecture to achieve slightly higher average accuracy, both methods protect well against catastrophic forgetting. Further, our experiments show that regularization of the task-conditioned hypernetwork outputs is crucial for avoiding catastrophic forgetting.

While highly capable, `HN-PPO` suffers from training instability. This instability is also present in the PPO baseline, but is amplified by training the same agent on multiple tasks, which results in very long total training times. When using default PPO hyperparameters for `HN-PPO`, gradient explosions randomly occurred in some experiments. Training was stabilized to an acceptable level after reducing the maximal gradient norm from 0.5 to 0.0001 (cf. Table 6). The choice of random seed also greatly influences the training outcome, which is another indicator of said instability. Additionally, the choice of training stop point is also of importance for good performance downstream, however, it remains unclear how to find the optimal stopping point under this aspect. Considering these limitations, reducing training instability should be an objective of further research on this subject. Understanding the causes for the variability in downstream performance could also significantly aid our line of work. Finally, the current method requires long training times of up to 1 day per task, which makes experiments with very long task sequences unpractical. Attempting to optimize `HN-PPO`'s convergence behavior and computational efficiency may therefore be of interest as well.

REFERENCES

Sayantan Auddy, Jakob Hollenstein, Matteo Saveriano, Antonio Rodríguez-Sánchez, and Justus Piater. Continual learning from demonstration of robotic skills, 2022. URL https://arxiv.org/abs/2202.06843.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Natalia Díaz-Rodríguez, Vincenzo Lomonaco, David Filliat, and Davide Maltoni. Don't forget, there is more than forgetting: new metrics for continual learning, 2018. URL https://arxiv.org/abs/1810.13166.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pp. 799–805. IEEE, 2021. doi: 10.1109/ICRA48506.2021.9560793. URL https://doi.org/10.1109/ICRA48506.2021.9560793.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives, 2020. URL https://arxiv.org/abs/2012.13490.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, mar 2017. doi: 10.1073/pnas.1611835114. URL https://doi.org/10.1073%2Fpnas.1611835114.

Ilya Kostrikov. PyTorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

Zhizhong Li and Derek Hoiem. Learning without forgetting. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam,*

The Netherlands, October 11-14, 2016, Proceedings, Part IV, volume 9908 of *Lecture Notes in Computer Science*, pp. 614–629. Springer, 2016. doi: 10.1007/978-3-319-46493-0_37. URL https://doi.org/10.1007/978-3-319-46493-0_37.

Nicolas Y. Masse, Gregory D. Grant, and David J. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018. doi: 10.1073/pnas.1803839115. URL https://www.pnas.org/doi/abs/10.1073/pnas.1803839115.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019. doi: 10.1016/j.neunet.2019.01.012. URL https://doi.org/10.1016/j.neunet.2019.01.012.

Matthew Riemer, Tim Klinger, Djallel Bouneffouf, and Michele Franceschini. Scalable recollections for continual lifelong learning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 1352–1359. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33011352. URL https://doi.org/10.1609/aaai.v33i01.33011352.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016. URL https://arxiv.org/abs/1606.04671.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Yusuke Urakami, Alec Hodgkinson, Casey Carlin, Randall Leu, Luca Rigazio, and Pieter Abbeel. DoorGym: A scalable door opening environment and baseline agent, 2019. URL https://arxiv.org/abs/1908.01887.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning, 2019. URL https://arxiv.org/abs/1904.07734.

Gido M. van de Ven, Tinne Tuytelaarsc, and Andreas S. Tolias. Three types of incremental learning (preprint), 2022. URL https://github.com/GMvandeVen/continual-learning.

Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJgwNerKvB.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3987–3995. PMLR, 2017. URL http://proceedings.mlr.press/v70/zenke17a.html.

A Appendix

A.1 Experiment Details

HN-PPO Hyperparameters Table 6 lists the hyperparameters used for experiments with `HN-PPO` and `HN-PPO+fc`. β and the task embedding dimension are specific to HN-PPO, the other were adapted from DoorGym's default values for PPO (Urakami et al., 2019).

Table 6: `HN-PPO` hyperparameters

Name	Value
Hypernetwork regularization β (Eq. 2)	0.001
Task embedding dimension	8
Learning rate	0.005
Parallel rollouts	8
PPO mini-batch size	256
PPO clipping parameter ϵ (Eq. 4)	0.3
Max. gradient norm	0.0001
Reward discount ratio γ	0.99
GAE lambda	0.95
Entropy bonus coeff. c_e (Eq. 5)	0.01
Value loss coeff. c_v (Eq. 5)	0.5

Network Architecture For PPO, the default DoorGym network architecture was reused: Both actor and critic networks are modeled as MLPs with 2 hidden layers of 64 neurons each. $tanh$ was used as the activation function. The critic network has an output dimension of 1 (the scalar value of the action), the actor has an output dimension of 6 (each controlling an input to the 6-DoF `floatinghook` robot). During training, a stochastic policy is used, with actions sampled from a normal distribution parameterized by the actor's output (mean) and a learned standard deviation vector. In evaluation, actions are taken directly from the actor's output, making the policy deterministic.

In `HN-PPO`, the target networks have the same architecture as laid out above for PPO. The hypernetwork controlling their weights is an MLP with 2 hidden layers of 640 neurons each. In the hypernetwork, ReLU activation is used for the hidden layers. The output layer of the hypernetwork is a multi-head (blue in Figure 6) with no non-linearity applied. Each head represents one tensor containing parameters of the target network, with output dimensions chosen accordingly. For the actor shown in Figure 6, 7 heads are generated: a weight and bias tensor for each layer, plus an additional standard deviation vector for the Gaussian stochasticity layer. Accordingly, the shown critic network requires 6 heads to parameterize.
For the "fresh critic" (`HN-PPO+fc`) architecture (cf. Section 5.2), the critic is not parameterized by the hypernetwork, but is a stand-alone MLP. Accordingly, the number of output heads in the hypernetwork is reduced.

A.2 Supplementary Information

Accuracy Matrices In this section, we show the train-test accuracy matrices, from which the CL metrics in Section 4.2 are calculated (Díaz-Rodríguez et al., 2018). For each training setup, we show the element-wise average matrix of 3 experiments. Matrix entry $A_{i,j}$ represents the opening rate for an agent at the end of its training for task i, evaluated on task j.

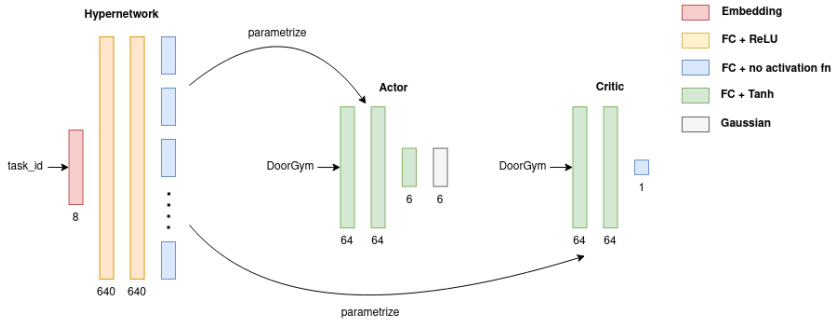


Figure 6: Network architecture in HN-PPO.

- PPO:

$$\begin{bmatrix} 1.0 & 0.21 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.48 & 1.0 & 0.0 & 0.0 & 0.0 & 0.03 \\ 0.11 & 0.01 & 0.32 & 0.0 & 0.57 & 0.0 \\ 0.0 & 0.29 & 0.0 & 0.32 & 0.0 & 0.62 \\ 0.0 & 0.01 & 0.0 & 0.0 & 0.96 & 0.0 \\ 0.01 & 0.01 & 0.0 & 0.0 & 0.0 & 0.63 \end{bmatrix}$$

- PPO-finetuning:

$$\begin{bmatrix} 1.0 & 0.21 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.61 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.24 & 0.0 & 0.0 & 0.53 & 0.0 \\ 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.68 \\ 0.19 & 0.06 & 0.27 & 0.0 & 0.37 & 0.0 \\ 0.01 & 0.05 & 0.0 & 0.0 & 0.0 & 0.36 \end{bmatrix}$$

- `HN-PPO`+fresh network

$$\begin{bmatrix} 0.98 & 0.39 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.18 & 0.97 & 0.0 & 0.0 & 0.01 & 0.0 \\ 0.23 & 0.0 & 0.27 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.02 & 0.0 & 0.25 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.02 & 0.0 & 0.59 & 0.0 \\ 0.0 & 0.08 & 0.0 & 0.03 & 0.0 & 0.69 \end{bmatrix}$$

- `HN-PPO`

$$\begin{bmatrix} 0.98 & 0.39 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.99 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.92 & 0.01 & 0.0 & 0.0 \\ 0.99 & 1.0 & 0.93 & 0.31 & 0.0 & 0.39 \\ 0.99 & 1.0 & 0.92 & 0.28 & 0.59 & 0.0 \\ 0.99 & 0.99 & 0.92 & 0.27 & 0.57 & 0.36 \end{bmatrix}$$

- `HN-PPO+fc`:

$$\begin{bmatrix} 1.0 & 0.45 & 0.0 & 0.0 & 0.01 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.69 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.69 & 0.36 & 0.0 & 0.16 \\ 1.0 & 0.99 & 0.7 & 0.34 & 0.25 & 0.0 \\ 1.0 & 1.0 & 0.68 & 0.31 & 0.25 & 0.01 \end{bmatrix}$$

- `HN-PPO+fc`, no regularization:

$$\begin{bmatrix} 1.0 & 0.45 & 0.0 & 0.0 & 0.01 & 0.0 \\ 0.66 & 1.0 & 0.0 & 0.0 & 0.0 & 0.01 \\ 0.33 & 0.0 & 0.44 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.03 & 0.0 & 0.62 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.83 & 0.0 \\ 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.04 \end{bmatrix}$$

Code availability The code used to run the experiments is available under `https://git.uibk.ac.at/csas8782/schoepf-bachelor-thesis`.