Final Report: Policy Adaptation during Deployment

Matthias Wolf
Technical University of Munich
matze.wolf@tum.de

Luca Eyring
Technical University of Munich
luca.eyring@tum.de

***Abstract*—Generalization across different environments is a significant challenge in reinforcement learning. Policy Adaption during Deployment (*PAD*) tackles this problem with the goal to adapt a trained policy to new environments without any prior data from them. Here, the key idea is to keep learning in a new environment which is realised through the introduction of a self-supervised task which does not require a reward signal. *PAD* can adapt well to visual changes in the test environment given a vision-based input. In this project we extend previous work by also investigating the performance of *PAD* when adapting the underlying dynamics of the test environment as well as when a state vector is used as input instead of an image. Our results suggest that *PAD* can help to adapt to physical changes in the environment and also performs well given a state vector input. However, *PAD* might also struggle to effectively adapt to certain modifications during deployment.**

I. Introduction

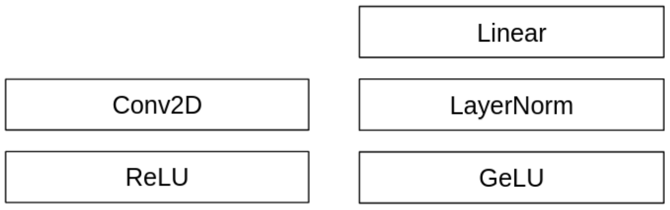
Policy Adaptation during Deployment (*PAD*) introduced in [1] is a novel approach in the area of reinforcement learning for robotics applications. It addresses the problem of the *sim2sim* or *sim2real* gap using a completely different approach than domain randomization. Generalization across multiple environments is known to be hard and tuning for the right amount of domain randomization is highly non-trivial. Therefore, [1] proposes the use of self-supervision to allow a feature extractor to continue training during deployment in a new environment.

To achieve this, an auxiliary task is added upon an intermediate representation of the policy network. While during training the standard reinforcement learning objective is trained jointly with the additional auxiliary task, during test-time solely the self-supervised objective keeps training the policy without needing any rewards.

This work is concerned with controlling robots in simulation, i.e., learning the correct control or action signals for certain robotic tasks.

II. Method

Hansen et al. [1] introduce two distinct auxiliary tasks. One of those, the inverse dynamics prediction, is used for learning control signals, which is what we want to achieve is this work. This task is learned by a self-supervised network that tries to predict the action a_{t-1} taken between the last state s_{t-1} and current state s_t at each time step. It can be applied to both discrete and continuous action spaces cast as either a classification or regression task.



(a) For vision input (b) For state vector input

Fig. 1: Encoder block

However, [1] solely uses vision input of either simulation environments or real-world scenes to demonstrate *PAD*. To further evaluate *PAD* we propose to replace the vision input by a state vector representation of the robot.

Herefore, we also change the architecture of the encoder to adjust for the change of 3D (RGB image) to 1D (state vector) input. In the original paper [1] the encoder consists of multiple encoder blocks that each contain a 2D convolutional layer followed by a ReLU. In our adapted encoder each block consists of a 1D linear layer followed by a LayerNorm [2] and a GeLU [3] as shown in figure 1.

III. Experiments

To empirically evaluate *PAD*, Hansen et al. [1] conduct a number of experiments on different environments where they apply vision-based perturbations to the test environment not seen during training. To further evaluate *PAD* we move away from solely permuting vision-based parameters like color changes and instead change parameters that impact the underlying dynamics of the given system. Additionally, we also evaluate *PAD* on a state vector input instead of just vision input.

In detail, we conducted experiments on the following 3 environments, which have already been implemented in the DeepMind Control Suite [4] using the MuJoCo library: cartpole, cheetah and walker (see figure 2). To implement our approach using state vectors, we use the observation vector from the DeepMind Control Suite as input. For each environment, we also implemented several adaptations to the original parameters of the environment to evaluate PAD in a *sim2sim* setting.

During deployment, we always compared the performance of a static policy to an updating policy using *PAD*. For each

(a) Cartpole

(b) Cheetah

(c) Walker

Fig. 2: DeepMind Control Suite environments

experiment we evaluate our trained model on 10 different seeds and report mean and standard deviation over them.

A. Cartpole

The first studied environment is the cartpole from [4], where the goal is to swing up a pole on a horizontally controlled cart from a downward to an upright position. The control is 1-dimensional, i.e., the force applied to the cart, while the observation is 5-dimensional. Rewarded are the degree of how centered the cart and how upright the pole is as well as a smaller angular velocity and control signal.

We first wanted to reproduce the paper results. Herefore, we used the same training setup as [1] and trained the model for 250k training steps. Afterwards, we evaluated the model on different training checkpoints.

We used 100 color-randomized environments like in [1] for evaluation in a *sim2sim* setting and were able to reproduce their results. As depicted in Figure 3, *PAD* needs sufficient training steps to outperform the baseline. *PAD* only outperforms the default policy after 100k training steps in our particular case.

For the cartpole swingup environment, we chose to permute the following parameters, each with 10 values that differ from the default environment:

- Pole length
- Pole radius
- Pole density

We compare the performance of *PAD* to a static policy in table I.

Cartpole	Environment	Default	*PAD*	Difference
Vision	Pole length	629.4 ±27.1	**653.6** ±15.5	+24.2
	Pole radius	720.9 ±15.7	**768.6** ±26.1	+47.7
	Pole density	**585.3** ±39.1	524.7 ±38.6	-60.6
State Vector	Pole length	512.4 ±38.5	512.4 ±31.5	+0.0
	Pole radius	875.9 ±0.3	877.6 ±0.5	+1.7
	Pole density	505.3 ±19.1	**516.3** ±33.2	+11.0

TABLE I: Results for the cartpole environment

For the vision setting, we can see that *PAD* can impressively adapt to changes to the pole length and radius. However, when we change the pole density during test time, *PAD* achieves worse results. Our hypothesis is that a change of density cannot be captured by the vision input of the encoder network, making "invisible" changes to the environment challenging for vision-based *PAD*.

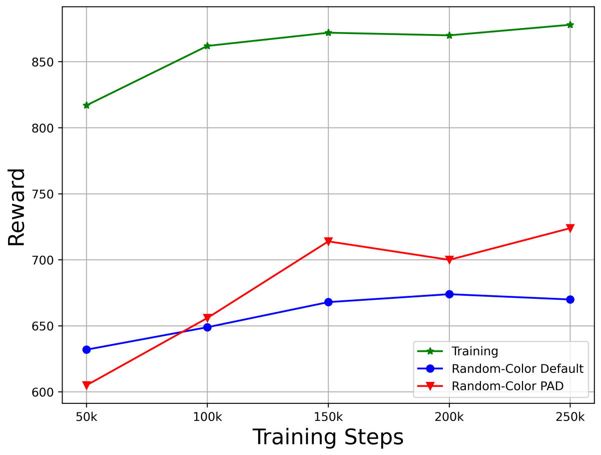


Fig. 3: Performance comparison of training checkpoints

For state vector input, we notice that the reward for the pole radius is as high as in the training setting. After further analysis, we realized that the mass does not actually change when we increase the pole radius in the DeepMind Control Suite. Thus, the pole radius represents only a vision-based perturbation. Hansen et al. already validated that *PAD* performs well in a solely vision adapted test environment and we can observe the same for this new vision perturbation giving the highest increase through the use of *PAD*. However, *PAD* also increases performance for physically changed environments by a small margin.

By studying the behavior of the learned policy from the state vector approach on video, we noticed that the movement patterns of the cartpole were extremely similar. To validate our new proposal, we therefore decided to conduct experiments on systems with more degrees of freedom, allowing for more "chaos" and variance in training data. The cheetah and walker environments that are presented in the following achieve this objective.

B. Cheetah

Compared to the cartpole, the cheetah environment has a lot more components and thus more degrees of freedom. Concretely, the control is 6-dimensional, representing the 6 controllable joints in the dynamical system, and the observation is 17-dimensional. The reward is purely based on the running speed of the cheetah.

As a *sim2sim* gap, we chose to permute the following parameters:

- Ground friction
- Cheetah mass

In table II we can see that *PAD* increases performance on the mass environment by a small margin, while it does not seem to be able to adapt to a change in ground friction. Here,

Cheetah	Environment	Default	*PAD*	Difference
Vision	Ground friction	258.6 ±6.0	256.9 ±6.7	-1.7
	Mass	215.7 ±26.1	220.4 ±21.5	+4.7
State Vector	Ground friction	269.5 ±28.5	269.3 ±18.3	-0.2
	Mass	340.1 ±27.3	**349.2** ±31.0	+9.1

TABLE II: Results on the cheetah environment

we observe a very similar trend in both vision and state vector input.

C. *Walker*

The walker is similar to the cheetah environment and consists of a 6-dimensional control representing the 6 controllable joints and a 24-dimensional observation. The reward consists of both running speed and upright posture of the walker.

This time, the following parameters of the environment were changed:

- Ground friction
- Torso length

Walker	Environment	Default	*PAD*	Difference
Vision	Ground friction	538.1 ±19.4	539.1 ±31.8	+1.0
	Torso length	270.2 ±17.6	**283.8** ±30.1	+13.6
State Vector	Ground friction	832.5 ±21.3	829.1 ±17.4	-3.4
	Torso length	744.0 ±27.9	**753.3** ±17.9	+9.3

TABLE III: Results on the walker environment

In table III we can see that again *PAD* does not seem to be able to adjust to the different levels of ground friction. However, *PAD* gives a performance increase for both vision and state vector in the changed torso length setting.

D. *Training environment*

In addition, we also evaluate both the vision and state vector approach on the unchanged training environment. Here, we observe a small decrease of performance from default deployment to using *PAD* as can be seen in table IV. This is inline with the experiments done by Hansen et al. [1]. The state vector approach outperforms the vision input in all 3 cases.

Train	Environment	Default	*PAD*	Difference
Vision	Cartpole	**870.4** ±0.5	859.5 ±0.8	-10.9
	Cheetah	252.6 ±5.2	247.9 ±7.8	-4.7
	Walker	**724.5** ±8.7	716.3 ±13.6	-8.2
State Vector	Cartpole	879.9 ±0.0	879.0 ±0.3	-0.9
	Cheetah	368.2 ±22.8	**374.4** ±24.1	+8.2
	Walker	**922.1** ±12.9	908.1 ±11.8	-14.0

TABLE IV: Results on the training environment

IV. Conclusion

As we can see from the presented results, *PAD* can indeed adapt to some changes in an environment and thus help to close the *sim2sim* gap. However, *PAD* seems to struggle with adapting to certain changes, such as perturbations to the ground friction in the cheetah and walker environments. We cannot offer a possible explanation for which cases *PAD* will offer an advantage and for which it does not, or even why that is the case. For now, the best way to find that out is empirically by evaluating *PAD* on the environment of interest.

As demonstrated in our experiments, changes to the environment do not necessarily have to be purely visual, but can also be modifications of parameters that influences the underlying dynamics of the system. Our experiments on the cheetah and walker environments also show that the state vector input for *PAD* leads to higher performance on systems with more degrees of freedom compared to vision based inputs. More chaotic environments improve variance in data during deployment and thus help *PAD*. In general, we believe that a state vector input generally improves performance compared to purely vision input since the latter tries to learn a sensible feature representation anyway which might be trained more easily by a state vector representation.

V. Future Work

An interesting extension of this work is certainly its application to a *sim2real* problem. There are however some aspects that need to be considered in that regard. An important aspect is certainly that an observation on a real robot is subject to sensor noise. Weather is suffices to simply feed noisy observations to *PAD* or it is worthwhile to model the uncertainty explicitly needs to be studied. Furthermore, it may be that some robot states are not easily observable in the real world, so there remains the question how valuable certain measurements are to the performance of *PAD*. This aspect is most likely highly dependent on the concrete robot though.

References

[1] N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang, "Self-supervised policy adaptation during deployment," *CoRR*, vol. abs/2007.04309, 2020. [Online]. Available: https://arxiv.org/abs/2007.04309

[2] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450

[3] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016. [Online]. Available: http://arxiv.org/abs/1606.08415

[4] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess, "dm_control: Software and tasks for continuous control," 2020.