Exploring modular architecture for Test Time Adaptation

Semester Project, Visual Intelligence for Transportation (VITA) Lab

Philippine des Courtils, Stella Petronio, supervised by Yuejiang Liu, Taylor Mordan
EPFL, Lausanne, Switzerland

June 16, 2022

***Abstract*—Inspired by human cognition, machine learning systems are gradually revealing the advantages of larger, modular architectures. Recent works show that not only do some modular architectures generalise well, but they also lead to better generalisation out of distribution, scalability properties, learning speed and interpretability. Our work explores modular architecture for Test-Time Adaptation (TTA). We have created a basic environment in which the performance of a so-called *reactive agent* is challenged and which in the future could serve as a basis for the evaluation of a so-called *proactive agent*. We provide a suitably designed model architecture in a causal model, combining it with Test-time Adaptation to achieve robustness in unseen environments. Our model architecture consists of a domain-generic and a domain-specific module, which deals with latent representation of movement dynamics. We illustrate the efficacy of our causal modelling and Test-Time adaptation by testing it on two robotic control tasks: the Cart-pole and the Walker-walk.**

I. Introduction

Deep learning through convolutions has reached tremendous predictive success including in reinforcement learning where agents learn actions from visual observations. However, current research in computer vision and visual-based reinforcement learning is still facing several challenges such as the robustness of test time prediction and the generalisation to unseen and especially out-of-distribution data [1], [2], [3], [4] the non-stationarity in the encountered environments [3], and the lack of interpretability of neural networks [1], [4], referred to as *black boxes*.

Deployment in an unseen environment was firstly addressed with prior anticipation of domain shifts at training time, by for instance learning a policy invariant to the environment. Recent works in computer vision and visual-based reinforcement learning [3], [2] considered instead adaptation without prior knowledge at test time, using self-supervised learning as a training signal as the reward or label signal was missing. We will be referring to agents performing *online adaptation* as *reactive* or *adapting* agents. Such agents can also deal with non-stationary environments when the changes come from the same or a smoothly changing distribution [2]. Moreover, the introduction of an auxiliary self-supervised task is in itself a step towards more interpretability.

In this present work, we've decided to tackle the problem in a slightly different way. We want to adapt a pre-trained model in an unseen environment without any reward, where some latent variables differ from the training environment, highlighting some limitations of a previously defined adapting agent [3]. We build our model upon the framework derived by Hansen et al. [3] and take a modularised approach into a *domain-specific* and a *domain-generic* module, as in Huang et al. [5]. Getting some inspiration from Hansen et al. [3] and Schölkopf et al. [1], we make the assumption that only a few parts of our network need to be updated at test time because we built a disentangled representation of the observed environment. Therefore, we adapt at test time the domain-specific module with a self-supervised loss as a training signal.

We carried out experiments on robotic tasks from DMControl suite [6]. We managed to highlight some limitations of the visual agent by Hansen et al. [3]. Through imitation learning training, we could validate some of our causal hypotheses. Finally, we started by investigating test time adaptation, and yielded encouraging results, though we need to stay critical regarding those as more sanity checks are required.

II. Related Work

Test-time adaptation in deep reinforcement learning addresses the fact that an agent might be facing an unseen environment during deployment phase.

The main takeaway of Hansen et al. [3] is to generalise the Test-Time Adaptation idea [2] to robotic and navigation tasks with visual input. Based on the fact that learning a robust policy invariant to all possible environment changes is unfeasible, the agent keeps on learning during deployment and adapt to its actual new environment. The second key idea of this work is to introduce a self-supervised auxiliary task to obtain a training signal since the reward signal is not available at test time. At training time, the model (Fig 1) jointly optimises a self-supervised objective and a reinforcement learning objective. Both sub-networks share a feature extractor. During the test phase, no reward signal is available. Therefore, only the self-supervised objective is optimised and the gradient from its loss is used to update the shared feature extractor. Consequently, the state representation is updated and the so-called *reactive agent* can deal with a distributional shift in the environment colours or a distracting object introduced in the scene.

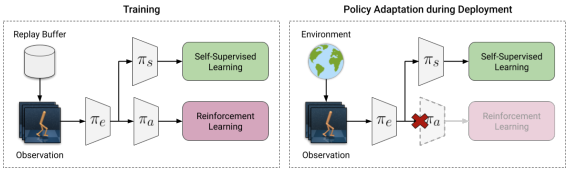

Fig. 1. Self-supervised policy adaptation network

Given suitable self-supervised tasks, Test-Time Adaptation improves performance under unseen environment conditions for both navigation (CRLMaze [7]) and motor (DMControl suite [8]) tasks. The choice of the self-supervised objectives has some impact on the performance gain, and we refer the reader to the original work by Hansen et al. [3] for more details. Those objectives are respectively rotation and inverse dynamics prediction for respectively navigation and robotic tasks. Finally, this adaptive method is easy to implement on top of any reinforcement learning algorithm.

However, visual-based reinforcement learning and Test-Time adaptation have some limitations. First of all, environment changes can be tackled up to some point. For instance, transitions in a non-stationary environment need to be smooth enough, and the agent fails for colours that are too far in the colour space from the training environment colour distributions. Moreover, an adapting agent reacts to changes with a delay. Secondly, the inverse dynamics task assumes constant dynamics. Therefore, at test time, the agent doesn't adapt well to changes in underlying dynamics which is an environment latent variable. We refer the reader to the section IV where we explore this flaw. Thirdly, and this limitation was also raised by Bodnar et al. [9], Hansen et al. [3] considered only an adaptation process to an episode, in which the environment remains unchanged or changes slowly. If we consider a long-term adaptation without reward, in which the agent encounters different environments and adapts to them, the agent experiences a decline in performance in all environments. This phenomenon is known as catastrophic forgetting *catastrophic forgetting*.

Latent variable modelling in multi agent reinforcement learning allows to go beyond some limitations of a reactive agent. Namely, inferring a latent variable from the environment and other agents can pave the way for the ego- agent to anticipate how other agents behaviours will change and to model the effect of its actions on the others' intentions. In a dual agent setting at train time, the works by Xie et al [10], Wang et al. [11] aim at modelling the opponent agent behaviour, inferring its *latent strategy*, namely its high level policy, and the corresponding *latent dynamics* according to which the strategy changes. Over repeated interactions with an opponent, the ego agent learns to model the opponent policy as a *latent strategy*. It then leverages these latent dynamics to *influence* [10] and *stabilise* [11] the opponent agent *strategy* and maximise the long term cumulative reward across interactions. Capturing latent dynamics through modularised and structured representation is relevant for our work. One major limitation though is that this method applies at train time only, in a setting where the ego agent experiences subsequent interactions and is able to train in between.

Causal modelling and modularisation could address some key research question about generalisation to out-of-distribution data, prediction robustness and transfer learning. Schölkopf et al. [1] give an in-depth analysis and formalisation of causal modelling, introducing *structural causal models* for machine learning application and relying on representation learning. One idea we got inspiration from is that causal models can be modularised in such a way that much of the causal structure should remain unchanged under different contexts or environments. Only a few modules may need to be updated. Making assumptions on a complex data generation system and designing the causal models carefully, such models are expected to generalise well under certain distributional shifts and to adapt faster to *interventions*, i.e. shifts in the distributions.

For one step more towards modularisation, Zhang et al. [5] proposed a principled framework for transfer reinforcement learning, called *AdaRL*, which learns a latent representation with a domain-shared and a domain-specific component between the source domains. *AdaRL* learns an optimal policy parameterised by the domain-specific parameters, and applies it to a new target domain. This is done without further optimisation of the policy, but only by estimating the values of the domain-specific parameters in the target domain, which can be achieved with little data from the latter.

III. Method

In this section, we underline and attempt to address some limitations of a reactive agent in a motor control task. Its architecture is strongly derived from the adapting agent architecture by Hansen et al. [3], and the self-supervised task is an inverse dynamics prediction one.

A. Baseline environment

To start with, the goal is to find an interpretable succession of domains more and more challenging for a *reactive agent* to cope with. By *domains*, we mean environments where the settings differ in a structured way from a visual or a non visual parameter. In future work, this set-up could become a baseline environment to compare a *reactive agent* to a so-called *proactive agent*. Such an agent would anticipate a change or shift before it occurs and adapt accordingly.

Our first idea is to make the adapting agent experience domain transitions such that at each transition, its performance drops and after some time rises up again. This would underline the limitations of a reactive behaviour and the need to anticipate a change. The second idea is to find successive domains in which the reactive agent performance breaks in a step-wise manner, and in which visual adaptation only is not sufficient. In the following section III-B, we will only consider changes in the dynamics of a robot control task and will justify our choice in section IV.

As we mentioned in the II section and as was reported by Bodnar et al. [9], the adapting agent is prone to catastrophic forgetting. Adapting on the fly leads to a decreased performance on the training environment after adaptation in an unseen environment. Therefore, we made the following artificial trick which should be improved later on. Whenever the domain changes, we initialise the agent weights with the pre-trained ones. Therefore, the *reactive agent* adapts again from scratch. One major disadvantage is that this implementation is heavy for an online adaptation, and that we cannot analyse the behaviour of the agent during transitions. However, we are only interested in the performance of the agent for a given domain, regardless of what happened before or will happened after. In this sense, we want to focus on stationary environments, but keep in mind that catastrophic forgetting must to be addressed to adapt to more complex non-stationary environments.

B. Imitation learning

Since our goal is to introduce a structured change in the environment, we must design our model accordingly. We start from the *Sparse Mechanism Shift* hypothesis introduced by Schölkopf et al. [1]. Small distribution -in our case domain- changes tend to manifest themselves in a sparse way within a causal disentangled model. We should therefore find an independent modularisation. Furthermore, this hypothesis implies that a causal model should adapt faster to *interventions*, i.e. domain shifts, that a purely predictive model and we would like our agent to generalise to unseen domains and extrapolate its experience across different environments.

Network Architecture. The network architecture was therefore designed assuming the relevance of two independent modules: a *domain-generic* (DG) module and a *domain-specific* (DS) module, represented respectively by the black and red modules in the Figures 2,3. The domain-generic module should capture the invariance across different domains, while the domain-specific module should be the only one affected by a domain change and needing to adapt at test time.

We turned the reinforcement learning (RL) problem into an *imitation learning* (IL) one, considering multi-source domains. This yields two major advantages: we don't need a reward signal from the environment and it simplifies the learning procedure across multiple domains. The key idea of this imitation learning algorithm stems from the fact that the agent learns a policy from demonstrations of experts agents trained on a specific domain. This boils down to a supervised learning setting, where the imitiation learning agents learns to predict the correct action.

The domain-generic module has an architecture very similar to the original Policy Adaptation during Deployment (PAD) approach [3]; the only difference in the current work subsists in the fact that the Reinforcement Learning module consists only of the *actor* network, without including also the critic network. This module actor head predicts the next action from current observation which is a classification task. Its self-supervised head learns an inverse dynamics task by predicting the action between two consecutive observations.

In this module, the reinforcement learning head π_{rl} with parameters θ_{rl} and the self-supervised (SS) π_{ss} with parameters θ_{ss} heads share a feature extractor network π_e with parameters θ_e. We obtain the following: $\pi(o;\theta) = \pi_{rl}(\pi_e(o))$ for the collection of parameters $\theta = (\theta_e, \theta_{rl})$ of a given policy network π , where o stands for an observation.

The domain-specific module deals with the environment *latent dynamics*, those dynamics being latent because they cannot be directly inferred from visual inputs and has parameters θ_{DS}. We considered two types of networks (see Fig. 2) which differ by their domain-specific (DS) modules. The first one (Fig. 2) takes as input to its domain-specific module the ground truth value of domain dynamics of interest, eg. the cart mass in a cart-pole swing-up task (See more on section IV-A). This first architecture choice could be perceived as cheating, but in fact it

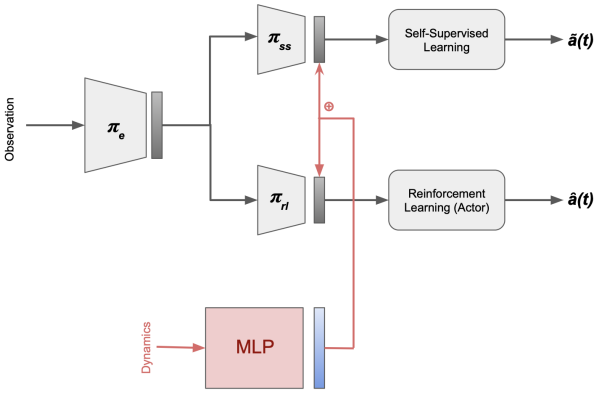

Fig. 2. DG and DS with GT vector as input

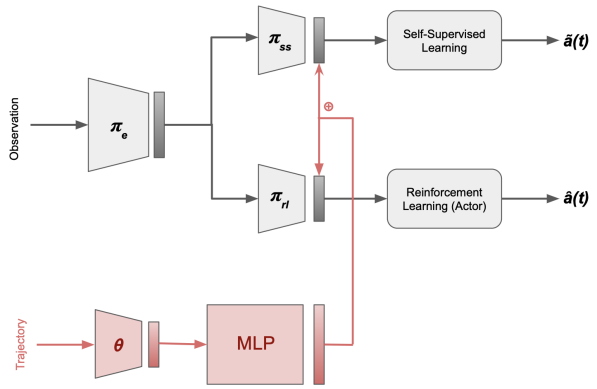

Fig. 3. DG and DS with short trajectory as input

would allow us to validate our modularisation hypothesis and present a baseline for the second architectural choice. The latter (Fig. 3) takes as input to its domain-specific module two successive transitions of observations and actions from an expert agent which hasn't been trained on the same domain. The idea is to infer dynamics in a self-supervised way from the observations of another agent's behaviour, independently from any environment metadata which won't be available at test-time.

Training and testing We train one imitation-learning agent per domain, optimising the domain-generic tasks on all the considered domains at the same time, and the domain-specific task on each domain separately. At training time, the policy, namely the reinforcement learning head, is jointly trained with the self-supervised objective. The input to self-supervised head is the output of π_e (as illustrated on Figures 2,3).

The *inverse dynamics* objective at train time is the following:

$$L_{SS}(\theta_e, \theta_{SS}, \theta_{DS}) = l(\tilde{a}(t), a(t)) \tag{1}$$

where l is the MSE-loss, $\tilde{a}$ the predicted action and a the ground truth action. The *actor* objective is quite similar:

$$L_{RL}(\theta_e, \theta_{RL}, \theta_{DS}) = l(\hat{a}(t), a(t)) \tag{2}$$

where l is again the MSE-loss, $\hat{a}$ the predicted action and a the ground-truth one. Therefore, the parameters of the shared feature extractor θ_e are updated with the inverse dynamics and the actor objective. Similarly, the domain-specific module parameters θ_{DS} are updated through the gradients from the self-supervised and the actor losses.

At test-time, we reuse the adaptation idea developed by Hansen et al. [3]. The self-supervised loss minimises the difference between the actions predicted by the self-supervised and the actor heads, and serves as a training signal for the domain-specific module. The two networks described in Fig. 2 are adapted differently. In the ground truth based model (2), the feature vector only is updated with the self-supervised gradient. In the trajectory input based network (3), parts of the domain-specific or the full module itself are updated with the self-supervised gradient.

Therefore, at test time, we have the following:

$$\theta_{DS}(t) = \theta_{DS}(t-1) - \nabla_{\theta_{DS}} L_{SS}(\theta_{SS}, \theta_e, \theta_{DS}(t-1)) \tag{3}$$

IV. Experiments

The experiments were performed on two tasks from the DMControl suite [6], namely the walker on task walk and the cart-pole on task swing-up. The original agents were trained for 500000 steps with the corresponding and respective framework described in [3].

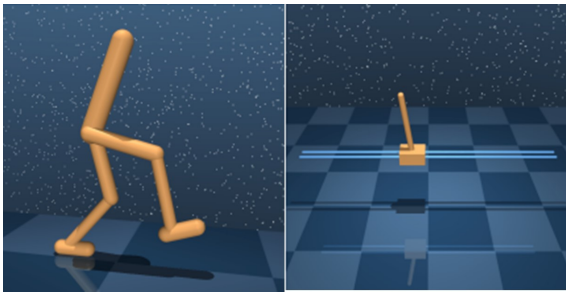

Fig. 4. walker-walk and cart-pole swing-up tasks

We therefore want to verify whether our causal hypothesis holds.

A. Baseline environment

We began by exploring how a reactive agent adapts to a change in the colour of the environment background. We experimented with different colour changes in the background, using colours gradually further from the training environment. However, we encountered two main issues. First of all, we observed a non-linear response in the adapting agent performance to changes in background colour. Secondly, quantifying how hard a colour change was isn't trivial. In a future work perspective, we realised that anticipating visual changes wasn't necessary, as those changes do not impact the task to solve and the agent could learn to merely ignore those changes.

Modifying the underlying movement dynamics in the DMControl suite environment turned out to be the correct choice for the reason that the agent cannot ignore this modification.

Experimental setup. We modify the cart-pole and the walker-walk environments in OpenAI Gym ([12]). Changes in state dynamics are considered, such as the change in the mass of the cart in cart-pole setting and the change in the force applied to the walker agent. The dynamics were modified every 300 timesteps and we averaged the agent performance on the defined environment during 100 episodes of 1000 steps. At each domain transition, the weights are initialised back to their pre-trained value, as explained in section III-A.

In order to find meaningful domains on which to subsequently train RL agents, we had to thoroughly explore the dynamics of the agents. In the case of the cart-pole setting, we decreased the cart mass from 1, which is the default value in DMControl ([6]), to 0.1 in a stepwise manner, in the training environment. For the walker-walk setting, we applied a force on the agent, exactly as shown in the lower part of Fig. 5. We started from a force of 0 magnitude, which is the default setting in the DMControl environment ([6]), decreasing it every 300 steps by 1 unit.

Results. We indeed observed a drop in average reward for the reactive agent from mass 0.4 for the cart-pole and force -1 for the walker. Figures 6 and 7 (in section VI) show the way we proceeded to pick up the right domains, i.e. by zooming in each time we notice a performance drop.

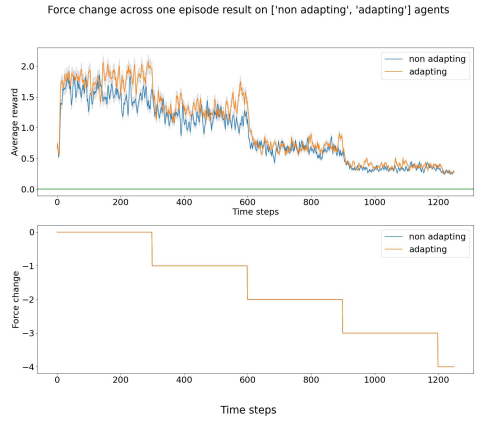

Fig. 5. Effect on force change on walker average reward and 95% confidence interval

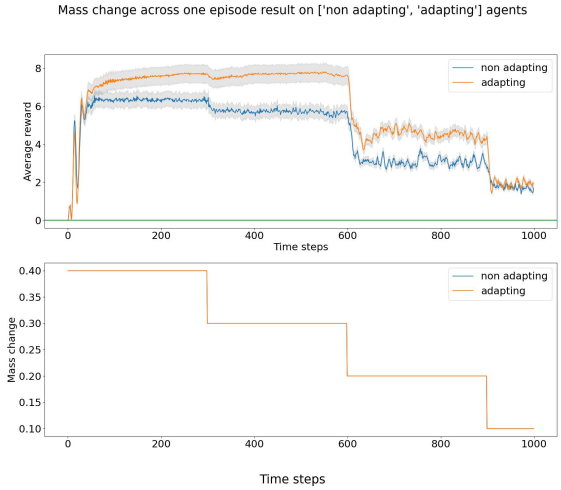

Fig. 6. Effect on mass change on cart-pole average reward and 95% confidence interval

After several experiments, we focused on the following four domains for the mass of the cart $[0.2, 0.25, 0.3, 0.4]$ (Figure 6) and three subsequent domains $[-1, -2, -3]$ (Figure 5) for the applied force on the walker-walk agent.

B. Imitation learning

Imitation learning framework. We implemented *behavioural cloning* (BC) imitation learning, where imitation learning agents are trained in a supervised fashion based on *experts* demonstrations. In this work, since behavioural cloning has some limitations, the *DAgger* algorithm ([13]) was applied and we refer the reader to the work by Attia et al. for more details.

We firstly trained the *experts* with reinforcement learning to achieve optimal performance for each *domain*. Then, we collected the *experts*'s trajectories. These trajectories were subsequently treated as i.i.d. state-action pairs, i.e. considering the state as input and the action as label. To build the imitation learning datasets for each specific domain, the *experts* were deployed in the training environment and collected input/output pairs. Finally, we trained each imitation learning agent as described below.

Experimental setup and training. Our imitation learning model consists of a domain-generic part shared by all imitation learning agents and a domain-specific part. At training time, the domain-generic module is updated with losses from the four agents simultaneously whereas the domain-specific module is trained independently.

As a first step, we build a domain-specific module which takes as the environment dynamics true value (Figure 2). This model is referred to as *IL with GT*, for ground truth.

As a second step, we wanted to test how powerful the domain-generic module is on its own, so we set up a model by removing the corresponding domain-specific module. In this case, one single agent was trained across all environments. This model is referred to as *IL with DG only*.

Instead of using ground truth values as input to the domain-specific module, we wanted to infer dynamics from observations and actions. As a third step, we build a domain-specific module which input consists of a short trajectory (T), i.e. of 2 consecutive transitions (i.e. $\{(o(t-3), a(t-3)), (o(t-2), a(t-2)), (o(t-1), a(t-1))\}$) to derive the vector of dynamic features (Figures 2, 3). This model is referred it as *IL with T*. Those transitions are not sampled from the learning agent itself but rather from a so called *reference agent*, which is the same for all the simultaneously learning agents. This agent was trained on another domain and evaluated on the domain of interest. Its policy stays constant during evaluation, and its data is collected to build those small trajectories. In either case, with ground truth or trajectory as input to the domain-specific module, the output of the latter is concatenated with the output of the shared encoder π_e of the Self-Supervised Policy Adaptation during Deployment agent [3], as input to the self-supervised and actor heads. (See Fig. 2, 3).

Network details. For the domain-generic module, observations are stacked frames ($k = 3$) rendered at 100×100 and cropped to 84×84, i.e. inputs to the network are of dimensions $9 \times 84 \times 84$, where the first dimension specifies the channel numbers and the succeeding ones stand for spatial dimensions. The feature extractor π_e is constituted of 8 convolutions layers and outputs features of size $32 \times 21 \times 21$ in DMControl. Both π_{rl} and π_{ss} are built as 3 convolutional layers followed by 3 fully-connected layers. The domain-specific module with ground truth as input is designed as a simple multilayer perceptron network (MLP) with 3 fully connected layers (see Fig. 2) of 20 units. The input shape is of size 2. The domain-specific module with trajectory is composed of an encoder θ whose task is to process the visual input. θ consists of 3 convolutional layers and has the same architecture as the full encoder $\pi_{ss}(\pi_e)$ or $\pi_{rl}(\pi_e)$. A multi-layer perceptron follows this encoder, and has the same architecture in both kinds of domain-specific modules, with an output shape of 8.

Results training. In the tables I, II we report the performance of the models described above for the cart-pole and walker-walk tasks across different domains. The cumulative rewards for each agent were measured across 100 roll-outs of the imitation learning policies, after 10 iterations of the DAgger algorithm. We report the mean and the standard deviation.

We firstly compare each imitation learning agent with the corresponding *baseline agent*. This agent corresponds to a non-adapting agent trained on default setting and makes up our lower performance bound (aside from the *IL with DG only* agent). We also compare performance to the *adapting agent* which is an adapting agent trained on default setting, but adapting according to Hansen et al. [3] method. The adapting agent performance should stay above the non-adapting one as long as the dynamics shift is not too far from the training domain conditions. Indeed, visual feedback can slightly compensate up to some point the disequilibrium induced by a change in dynamics. But it is

not enough, as the self-supervised head assumes unchanged dynamics.

As shown in tables below, the domain-generic alone consistently under-performs *IL with GT* and *IL with T* which emphasises the need for the domain-specific module in our network architecture, as we expected. Moreover, we can underline that the performances reached by *IL with T* agents are very close to those obtained with *IL with GT* agents. This means that we can infer dynamics from observations only, without any label or signal from the deployment environment.

Cart-pole				
Agent/Mass	**0.2**	**0.25**	**0.3**	**0.4**
Baseline	399 +/- 60	470 +/- 70	565 +/- 47	664 +/- 37
Adapting	446 +/- 70	541 +/- 67	614 +/- 59	744 +/- 65
Expert	936 +/- 1	917 +/- 5	924 +/- 2	924 +/- 2
IL with DG only	90 +/- 1	94 +/- 1	110 +/- 2	153 +/- 2
IL with GT	929 +/- 1	818 +/- 42	900 +/- 25	812 +/- 34
IL with T	929 +/- 10	901 +/- 82	870 +/- 3	898 +/- 3

TABLE I
MEAN AND STANDARD DEVIATION FOR CUMULATIVE REWARD ACROSS ONE EPISODE FOR CART-POLE SWING-UP TASK

Walker-walk			
Agent/Force	**-1**	**-2**	**-3**
Baseline	909 +/- 40	737 +/- 49	583 +/- 52
Adapting	891 +/- 60	727 +/- 74	568 +/- 61
Expert	942 +/- 23	937 +/- 26	932 +/- 32
IL with DG only	411 +/- 116	505 +/- 174	605 +/- 152
IL with GT	914 +/- 81	921 +/- 65	933 +/- 28
IL with T	937 +/- 28	916 +/- 89	916 +/- 40

TABLE II
MEAN AND STANDARD DEVIATION FOR CUMULATIVE REWARD ACROSS ONE EPISODE FOR WALKER-WALK TASK

Experimental set-up for testing. Due to time constraint, we could only verify our framework on imitation learning agents with ground truth input, which is the simplest setting (see Fig 2). At test-time, we adapted only the *domain-specific* feature vector (in blue on the schematic). This feature vector is the output of the multi-layer perceptron, which is concatenated with the processed observations as an input to the actor and the self-supervised head. The agents we considered were trained on domain of cart mass 0.4 and on domain of applied force -1 for respectively cart-pole swing-up and walker walk tasks. Since the ground truth value is not available at test time, we initialised the feature vectors with the ground truth value of their respective training domain, i.e. 0.4 and -1. We evaluated those agents on in-range and out-of-range domains, screening for the best learning rates on 5 seeds and 100 policy rollouts per seed. As a reminder, our range domains are $[0.2, 0.25, 0.30.4]$ for the cart mass and $[-1, -2, -3]$ for the applied force. The default parameters are 1 for the cart mass and 0 for the applied force. We tested agents with in-range values, namely a mass of 0.25 and a force of -2, which are within the interpolation range of the training domains, and with out-of-domain values, namely 0.1 and -3.5.

Results testing. Below are presented the results mean and standard deviation for the best learning rates. For the cart-pole experiments in Tab.III, the results were obtained after evaluation on 374 time steps to stress the difference between adapting and non adapting agent, whereas for the walker experiments, those results in Tab. IV-B were obtained on 250 steps.

We can notice that adapting agent reaches significantly higher performance than non-adapting agent for out-of-range domains, i.e. for cart mass 0.1 and force -3.5. However, for in-range values, this difference cannot be clearly stated. This result stands in favour of our causal hypothesis, where the domain-specific module only needs to be updated at test time to catch up the domain shift. We would need though more sanity checks to fully understand the agent response to a change in the environment, as many more combinations of testing can be performed. Our hypothesis is that the structural relationship between the domains hasn't been fully captured by our network. In a future work, we could design more carefully our domain-specific module to enforce this structural relationship.

Cart-pole		
Test domain and learning rate \Agent	**Adapting agent**	**Non adapting agent**
Test domain 0.25, lr 0.1	2634 +/- 292	2417 +/- 133
Test domain 0.1, lr 0.1	1484 +/- 414	1157 +/- 195

TABLE III
CUMULATIVE REWARD MEAN AND STANDARD DEVIATION FOR CART-POLE SWING-UP TASK

Walker-walk		
Test domain and learning rate \Agent	**Adapting agent**	**Non adapting agent**
Test domain -2, lr 0.01	868 +/- 58	868 +/- 80
Test domain -3.5, lr 0.1	695 +/- 103	616 +/- 66

TABLE IV
CUMULATIVE REWARD MEAN AND STANDARD DEVIATION FOR WALKER WALK TASK

V. CONCLUSIONS AND FUTURE WORK

In this work, we highlighted some limitations of a reactive agent in a motor control task and we settled some milestones toward a more robust test-time adaptation, based on causal representation and imitation learning. A basic environment in which to test the performance of the reactive agent and evaluate its performance against that of the proactive agent was implemented. In addition to the work of Hansen et al. [3], we built our model with a modularisation based on two independent modules: a domain-generic and a domain-specific. The experiments results showed that both modules were relevant and adapting the domain-specific module at test time can improve performance on unseen domains. In future work, we could first enforce the structural relationship between domains because we believe it hasn't been fully captured by the domain specific. Secondly, we should implement test-time adaptation for the visual base agent. Moreover, we should explore an alternative way to handle catastrophic forgetting as we got rid of it artificially by reloading the weights at each change. Solving the challenge raised by catastrophic forgetting will help to include non-stationarity among environments. Finally, an exciting next step could be to extend our model to other types of dynamics changes and other robotic tasks in DMControl.

REFERENCES

[1] B. Schölkopf et al. "Toward Causal Representation Learning". In: *Proceedings of the IEEE* 109.5 (2021), pp. 612–634. DOI: 10.1109/JPROC.2021.3058954. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9363924.

[2] Yu Sun et al. "Test-Time Training with Self-Supervision for Generalization under Distribution Shifts". In: *ICML*. 2020.

[3] Nicklas Hansen et al. "Self-Supervised Policy Adaptation during Deployment". In: *CoRR* abs/2007.04309 (2020). arXiv: 2007.04309. URL: https://arxiv.org/abs/2007.04309.

[4] Sarthak Mittal, Yoshua Bengio, and Guillaume Lajoie. "Is a Modular Architecture Enough?" In: (2022). DOI: 10.48550/ARXIV.2206.02713. URL: https://arxiv.org/abs/2206.02713.

[5] Biwei Huang et al. “AdaRL: What, Where, and How to Adapt in Transfer Reinforcement Learning”. In: *arXiv preprint arXiv:2107.02729* (2021).
[6] Yuval Tassa et al. “Deepmind control suite”. In: *arXiv preprint arXiv:1801.00690* (2018).
[7] Vincenzo Lomonaco et al. “Continual Reinforcement Learning in 3D Non-stationary Environments”. In: *arXiv preprint arXiv:1905.10112* (2019).
[8] Saran Tunyasuvunakool et al. “dm-control: Software and tasks for continuous control”. In: *Software Impacts* 6 (2020), p. 100022. ISSN: 2665-9638. DOI: https://doi.org/10.1016/j.simpa.2020.100022. URL: https://www.sciencedirect.com/science/article/pii/S2665963820300099.
[9] Cristian Bodnar et al. “A Geometric Perspective on Self-Supervised Policy Adaptation”. In: (Nov. 2020).
[10] Annie Xie et al. “Learning latent representations to influence multi-agent interaction”. In: *arXiv preprint arXiv:2011.06619* (2020).
[11] Woodrow Zhouyuan Wang et al. “Influencing towards stable multi-agent interactions”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 1132–1143.
[12] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
[13] Alexandre Attia and Sharone Dayan. “Global overview of imitation learning”. In: *arXiv preprint arXiv:1801.06503* (2018).

VI. Appendix

A. Baseline environment

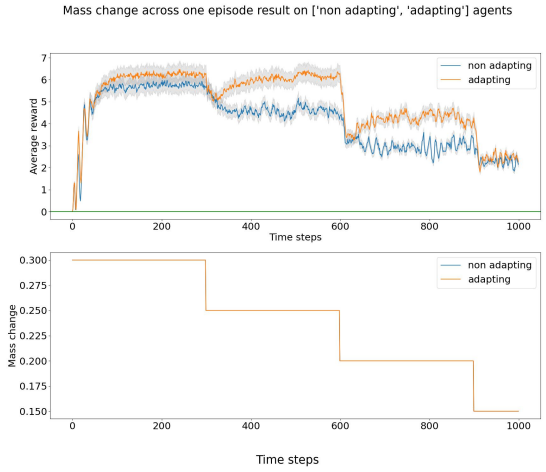

Fig. 7. Effect on mass change on cart-pole average reward and 95% confidence interval - closer look

B. Imitation learning evaluation

The following experiments were run on 5 seeds and 100 policy rollouts. We present here the full learning rate screening experiments along with the episodic reward curves.

Cart-pole						
Agent \Learning rate	**0.0001**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**
Adapting agent	204 +/- 35	203 +/- 35	203 +/- 34	201 +/- 32	205 +/- 33	223 +/- 49
Non adapting agent	205 +/- 36	205 +/- 37	205 +/- 33	203 +/- 34	206 +/- 37	207 +/- 34

TABLE V
Test-time adaptation learning rate screening for Cart-pole swing-up on domain 0.1

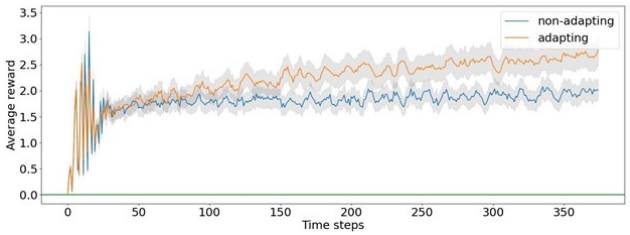

Fig. 8. Test-time adaptation for domain 0.1 (Cart-pole task), average reward and 95% confidence interval

Cart-pole						
Agent \Learning rate	**0.0001**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**
Adapting agent	679 +/- 67	678 +/- 72	679 +/- 67	682 +-/ 73	725 +/- 75	742 +7- 84
Non adapting agent	680 +/- 62	685 +/- 57	681 +/- 58	680 +/- 61	676 +/- 62	680 +/- 58

TABLE VI
Test-time adaptation learning rate screening for Cart-pole swing-up on domain 0.25

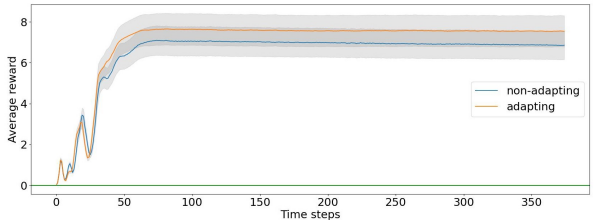

Fig. 9. Test-time adaptation for domain 0.25 (Cart-pole task), average reward and 95% confidence interval

Walker-walk						
Agent \Learning rate	**0.0001**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**
Adapting agent	627 +/- 65	626 +/- 64	626 +/- 64	621 +/- 61	647 +/- 87	695 +/- 103
Non adapting agent	616 +/- 66	616 +/- 66	616 +/- 66	616 +/- 66	616 +/- 66	616 +/- 66

TABLE VII
Test-time adaptation learning rate screening for walker walk on domain -3.5

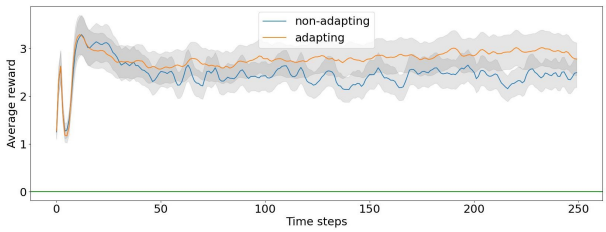

Fig. 10. Test-time adaptation for domain -3.5 (Walker-walk task) and learning rate 0.1, average reward and 95% confidence interval

Walker-walk						
Agent \Learning rate	**0.0001**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**
Adapting agent	863 +/- 67	869 +/- 59	868 +/- 58	868 +/- 58	854 +/- 60	814 +/- 120
Non adapting agent	868 +/- 81	868 +/- 81	867 +/- 80	868 +/- 80	868 +/- 80	868 +/- 80

TABLE VIII
Test-time adaptation learning rate screening for walker-walk on domain -2