COMPARING AUXILIARY TASKS FOR LEARNING REPRESENTATIONS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning state representations has gained steady popularity in reinforcement learning (RL) due to its potential to improve both sample efficiency and returns on many environments. A straightforward and efficient method is to generate representations with a distinct neural network trained on an auxiliary task, i.e. a task that differs from the actual RL task. While a whole range of such auxiliary tasks has been proposed in the literature, a comparison on typical continuous control benchmark environments is computationally expensive and has, to the best of our knowledge, not been performed before. This paper presents such a comparison of common auxiliary tasks, based on hundreds of agents trained with state-of-the-art off-policy RL algorithms. We compare possible improvements in both sample efficiency and returns for environments ranging from simple pendulum to a complex simulated robotics task. Our findings show that representation learning with auxiliary tasks is beneficial for environments of higher dimension and complexity, and that learning environment dynamics is preferable to predicting rewards. We believe these insights will enable other researchers to make more informed decisions on how to utilize representation learning for their specific problem.

1 INTRODUCTION

In reinforcement learning (RL), the often complex interplay of observations, actions and rewards means that algorithms are often sample-inefficient or cannot solve problems altogether. State representation learning tackles this issue by making information encoded in observations, and possibly actions, more accessible. Mnih et al. (2013) first introduced deep RL to extract information from the high-dimensional observations provided by Atari games. Later, Munk et al. (2016) and then Stooke et al. (2020) made a case for decoupling representation learning from solving the RL task of maximizing cumulative rewards. As decoupling amounts to using a separate neural network (or other method) to calculate representations, we will call this explicit, rather than implicit, representation learning – although the distinction is not always so clear, for instance when a number of layers are merely furnished with different prediction heads. Munk et al. (2016) additionally introduce predictive priors, learning targets that differ from the RL task but are also based on data generated by the environment. Other authors such as Legenstein et al. (2010) Wahlström et al. (2015), Anderson et al. (2015) and Shelhamer et al. (2017) have proposed further learning targets and started to call these *auxiliary tasks*. Not all the listed works learn explicit representations, although doing so comes with advantages: Representations generated by separate networks can replace raw observations and, optionally, actions as inputs to an RL algorithm. The individual parts of such agents then have distinct purposes. Different tasks do not interfere with each other, and representations can remain agnostic to the RL task. Individual components can easily be exchanged, making this approach flexible.

Most of the works above learn representations for discrete control on environments with high-dimensional visual observations. Recently, it has been shown that representation learning can improve returns also on non-visual continuous control benchmark problems of RL, such as the MuJoCo control tasks (Ota et al. (2020)). As far as we know, a comparative study of auxiliary tasks on these continuous control benchmark environments has not been done before, despite their importance to RL research. This paper presents such a study of auxiliary tasks. We hope it provides other researchers with an unbiased and, within the limits of our computational restraints, broad comparison that can assist them in selecting auxiliary tasks for representation learning in the context of their research problems.

We conduct our comparison by investigating returns and sample efficiencies achieved with common auxiliary tasks on five diverse environments. These environments cover a range of observation and action dimensionalities, and varying levels of complexity concerning the relationship between observations, actions and rewards. Explicit representations are computed with OFENet (Ota et al. (2020)), and used as inputs to the off-policy RL algorithms TD3 and SAC. Since one environment, FetchSlideDense-v1, cannot be solved with baseline TD3 or SAC, we additionally train agents with hindsight experience replay (HER). Our results show that representation learning with auxiliary tasks significantly increases both maximum returns and sample efficiency for high-dimensional and complex environments, although it has little effect on simpler, smaller environments. We find that learning representations based on environment dynamics, for instance by predicting the next observation, is superior to using reward prediction. Interestingly, adding representation learning to TD3 makes it possible to train somewhat successful agents on the FetchSlideDense-v1 environment, even if baseline TD3 cannot learn anything at all.

2 Related work

Many works in recent years have made use of some auxiliary task to learn state representations. We cite multiple of these in Section 3. Ota et al. (2020) for instance, whose representation learning network we use here, predict the next observation from current observation and action. There are however few papers which compare auxiliary tasks to each other. Lesort et al. (2018) have written a thorough survey of state representation learning which summarizes different auxiliary tasks and includes a comprehensive list of publications. It is however a purely theoretical discussion of methods without any empirical comparisons or results. There are two empirical comparisons of auxiliary tasks (Shelhamer et al. (2017) and de Bruin et al. (2018)), which differ in various aspects from ours. Shelhamer et al. (2017), like us, compare auxiliary tasks on various environments. In contrast to us, they use Atari games with visual observations. Another difference is that Shelhamer et al. (2017) don't fully decouple representation learning from the RL algorithm. Instead, they merely connect a different prediction head to train the initial, convolutional part of the deep RL algorithm on auxiliary targets. Their results generally vary across environments. An interesting feature of their paper is the comparison of individual auxiliary tasks to a combination of several. Curiously, the combination never clearly outperforms the respective best individual tasks. The second comparison, de Bruin et al. (2018), uses only one car race environment but with several race tracks. It provides multimodal observations which again include of visual data. In contrast to our approach, loss functions of auxiliary tasks and RL task are linearly combined, and auxiliary tasks are investigated by removing their individual loss terms from the combination. Representations are hence learned implicitly.

3 Auxiliary tasks

In this section we present common auxiliary tasks, of which we will empirically compare the first three while the last two do not work with our setup. An overview of the tasks is presented in Figure 1. To discuss these tasks, we first need to briefly formalize the reinforcement learning problem: An environment provides reward r_t and observation o_t at time step t. The agent then performs an action a_t, which generates a reward r_{t+1} and leads to the next observation o_{t+1}. This cycle is modeled by a Markov decision process, which means that there can be randomness in the transition from o_t to o_{t+1}, given some a_t. The Markov property implies that o_{t+1} only depends on o_t and a_t which already contain all past information. It does not depend on previous states or actions. The goal of the RL agent is to maximize cumulative reward (return). Altogether, these components are the ones available to auxiliary tasks, and various possible combinations are used.

Reward prediction (*rwp*) is the task of predicting r_{t+1} from o_t and a_t. Works that use *rwp* include Munk et al. (2016), Jaderberg et al. (2016), Shelhamer et al. (2017), Oh et al. (2017) and Hlynsson & Wiskott (2021). With explicit representation learning, *rwp* is limited in that it can only be applied to environments which provide non-trivial rewards. Representations might otherwise become decoupled from o_t and a_t if r_{t+1} is constant. A RL algorithm trained on these representations could not learn anything at all. It can be argued that representations based on reward prediction have an advantage over those learned with other auxiliary tasks as they are optimized towards the actual RL task. On the other hand, *rwp* is somewhat redundant to the RL task of maximizing returns, although it only considers the immediate next reward and is therefore less noisy (Shelhamer et al. (2017)).

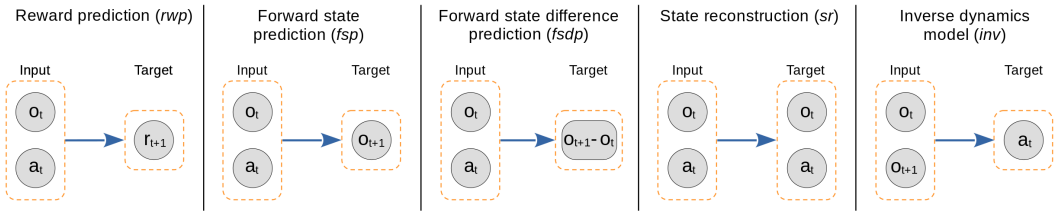

Figure 1: An overview of inputs and prediction targets of common auxiliary tasks.

Forward state prediction (*fsp*) is the task of predicting o_{t+1} from o_t and a_t. It is a popular task and used e.g. by Wahlström et al. (2015), Munk et al. (2016), van Hoof et al. (2016), Pathak et al. (2017) and Ota et al. (2020). In contrast to our work, several of these deal with high-dimensional image observations and therefore try to predict the next representation rather than the next observation in time. The *fsp* task, as opposed to *rwp*, can be applied to any kind of environment without conditions. Its task amounts to learning environment dynamics as it is done e.g. in model-based RL. The *fsp* task can thus be considered model-based RL, although we combine it with model-free RL algorithms.

Forward state difference prediction (*fsdp*) describes the task of predicting $(o_{t+1} - o_t)$ from o_t and a_t. Anderson et al. (2015) and Jaderberg et al. (2016) use *fsdp*. Conceptually, it is very similar to *fsp*. While *fsdp* also learns environment dynamics, Anderson et al. (2015) claim that successive observations are very similar and predicting only the difference thus gives more explicit insight into environment dynamics. The *fsdp* task requires a notion of difference, though this is not a practical issue as observations are usually encoded as numerical vectors. In comparison with *fsp*, the *fsdp* task should provide an advantage in environments without excessive noise or stark changes between successive observations. This would make *fsp* more robust, but *fsdp* particularly suited for environments simulating real-world physics.

State reconstruction (*sr*) (used e.g. in Jaderberg et al. (2016) and Shelhamer et al. (2017)) is the task of reconstructing o_t (and possibly a_t) from o_t and a_t. This is the classical autoencoder task, but does not make sense in our setup where data dimensionality is expanded by concatenation. Our representations thus always contain the raw o_t and a_t, and reconstruction would amount to simply filtering these out. No useful representations could be learned. In general, *sr* is related to *fsp* but, crucially, does not learn environment dynamics. It therefore seems reasonable to assume that *fsp* will in most cases be a better choice for learning representations for RL. The results of both Jaderberg et al. (2016) and Shelhamer et al. (2017) confirm this.

The **inverse dynamics model (*inv*)**, framed as a learning task, predicts a_t from o_t and o_{t+1}. Works using this task include Shelhamer et al. (2017) and Pathak et al. (2017). While *fsp* and *fsdp* focus on learning transition probabilities of the environment, the *inv* task considers how actions of the agent correlate with changes in the environment. We cannot use the *inv* task here because learning these representations is not possible in conjunction with actor-critic algorithms: Gradients would have to pass through the – usually not differentiable – environment in order to be propagated back from critic to actor. Section A.1 in the appendix provides a detailed explanation.

Various **other priors** have been proposed by different authors (for a list, see Lesort et al. (2018)). Noteworthy examples include the slowness principle used in Legenstein et al. (2010) and the robotics prior by Jonschkowski & Brock (2015). However, these are not as commonly used as the previously mentioned tasks and some of them are even specific to certain problem settings. We thus exclude them from our comparison.

In addition to the tasks above, there are several works on combining auxiliary tasks. A popular combination is *fsp* or *fsdp* with *rwp* (e.g. Munk et al. (2016), Jaderberg et al. (2016)), various others exist. Lin et al. (2019) have even proposed a method to adaptively weigh different auxiliary tasks.

4 METHODS

This section explains the neural network we use to learn representations with the auxiliary tasks, the RL algorithms we train on these representations and the environments we use for training.

4.1 REPRESENTATION LEARNING NETWORK

To train explicit representations on auxiliary tasks, we use the network architecture of OFENet by Ota et al. (2020). The architecture is composed of two parts. Its first part calculates a representation z_{o_t} of o_t, and the second part calculates a representation z_{o_t,a_t} of z_{o_t} and a_t. Internally, the parts stack MLP-DenseNet blocks which consist of fully connected and concatenation layers. The whole arrangement is visualized in Figure 2. For our experiments we give both parts of OFENet the same internal structure (apart from input dimensionality), but adjust parameters to different environments as described in Table 1 in the appendix. The auxiliary loss is calculated as the mean squared error between predicted and actual target.

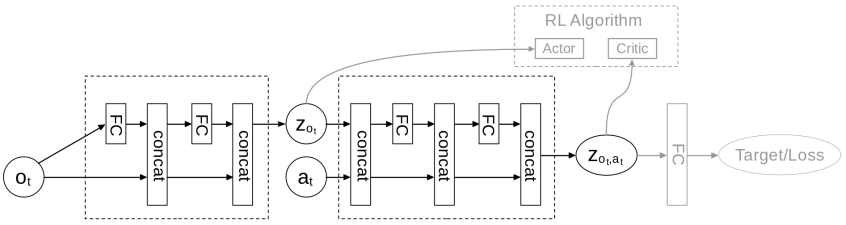

Figure 2: Sketch of the OFENet architecture, modified from Ota et al. (2020). Observation o_t and action a_t are used to calculate representations z_{o_t} and z_{o_t,a_t}. These, in turn, are passed into the RL algorithm (light grey). The prediction target necessary to evaluate the auxiliary loss is calculated with one fully connected layer (FC, light grey) from z_{o_t,a_t}.

OFENet is a good choice for comparing auxiliary tasks as it is a rather generic architecture for learning representations of expanded dimensionality. Besides OFENet, we are not aware of any other approaches in the field of RL to learn representations without dimensionality reduction. Most works use (variational) autoencoders, which have been shown to be very powerful especially for image data. An advantage of the dimensionality expansion approach of OFENet, however, is that it can be applied to smaller, simpler environments. In the context of a comparison, this might eliminate some excess factors of variation potentially introduced by more complex network architectures. As a matter of fact, Ota et al. (2020) show that their dimensionality expansion approach can even be successful for environments of somewhat high dimensionality such as Humanoid-v2 with 292-dimensional observations. We make use of that in our study.

4.2 REINFORCEMENT LEARNING ALGORITHMS

To solve the RL task of maximizing returns, we use TD3 (Fujimoto et al. (2018)) and SAC (Haarnoja et al. (2018)), two well-known state-of-the-art RL algorithms. They are both model-free off-policy actor-critic methods. Comparing auxiliary tasks against these two presents a trade-off between the computational expense of training all agents required for our comparison (hundreds per RL algorithm) and investigating more than one algorithm to avoid results being biased. We chose these two algorithms in particular because they are powerful and also quite popular, which makes them a testbed that is both non-trivial and particularly relevant to potential readers.

We study one environment, FetchSlideDense-v1, which is too difficult to solve with baseline TD3 and SAC. It does however become at least partially solvable when adding hindsight experience replay, first proposed by Andrychowicz et al. (2017). HER infuses the replay buffer used by off-policy algorithms with additional samples copied from previous episodes. In these copied samples, it changes the reward signal to pretend the agent had performed well, in order to present it with positive learning signals. Additional supposedly successful episodes provide a stronger incentive for the agent to learn, which makes learning in complex environments easier. Nowadays it is widespread practice to use HER for robotics tasks such as FetchSlideDense-v1.

4.3 ENVIRONMENTS

We perform our study on five different environments: A simulated pendulum, three MuJoCo control tasks and a simulated robotics arm. They span a large range of size and complexity. Size, here,

refers to the dimensionality of observation and action space, while complexity is about how difficult it is to learn a sufficient mapping between observation space, action space, and rewards. The three MuJoCo control tasks differ in size but are controlled by similar dynamics, which allows for a very direct comparison. All studied environments are depicted in Figure 3. Sizes of observation and action spaces, and of the corresponding representations learned with OFENet, are listed in Table 1 in the appendix.

In the following, all five environments we use are briefly described. For further details on the first four, we refer the reader to the documentation of OpenAI Gym (Brockman et al. (2016)).

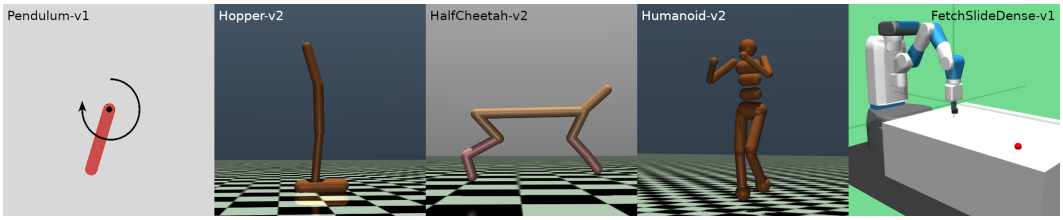

Figure 3: Sample images rendered to visualize the environments. The image of FetchSlideDense-v1 is from Plappert et al. (2018).

Pendulum-v1 is a simple and small classic control environment in which a pendulum needs to be swung upwards and then balanced in this position by applying torque. Its observations quantify angle and angular velocity of the pendulum. The reward at each time step is an inversely linear function of how much the angle differs from the desired goal, how much the angle changes, and how much torque is applied.

Hopper-v2 is the first of three MuJoCo control tasks we consider here. It is based on a physical simulation of a two-dimensional single leg with four parts, which can be controlled by applying torque to the three connecting joints. This makes it comparatively small and simple as well. The observation contains some of the angles and positions of parts and joints, as well as their velocities. The reward at a given time step mostly depends on how much the hopper has managed to move forward, plus a constant term if it has not collapsed.

HalfCheetah-v2 is another two-dimensional MuJoCo control task, similar to Hopper-v2 but larger and more complex. It already consists of 9 links and 8 joints, with action and observation space similar in nature to those of Hopper but consequently of larger dimension. The reward is again based on how much the HalfCheetah has moved forward since the last time step.

Humanoid-v2 is the last MuJoCo control task we use in our comparison. As opposed to the previous two, it is three-dimensional. It roughly models a human, which leads to actions and observations that are similar to those of Hopper-v2 and HalfCheetah-v2, but of far higher dimension. In comparison, Humanoid-v2 is a very large environment. The reward is once again primarily based on forward movement plus a constant term if the robot has not fallen over.

FetchSlideDense-v1 (Plappert et al. (2018)) is an environment which incorporates a simulated robotics task. It is not much larger than HalfCheetah-v2, but much more complex than any of the other tasks. A three-dimensional arm needs to push a puck across a low-friction table so that it slides to a randomly sampled goal position which is out of reach of the arm. The action controls where to move the tip of the arm, while the observation encodes position and velocities of arm and puck as well as the position of the goal. The reward is the negative distance between puck and goal position, which makes it constant as long as the arm does not hit the puck. In their technical report, the authors show that this task is very difficult to solve even with state-of-the-art methods, unless additional methods such as HER are deployed. Instead of return, FetchSlideDense-v1 is evaluated by success rate, which describes in how many cases out of 100 the puck ended up close to its goal.

5 Experiments

To compare the auxiliary tasks, we train agents with baseline TD3 and SAC on raw observations (baseline) and on representations learned with auxiliary tasks. We do this for each of the five en-

vironments. Additionally, for FetchSlideDense-v1, we combine TD3 and SAC with HER and train these on raw observations as well as on representations learned with auxiliary tasks. All of the aforementioned experiments are conducted five times with the same set of random seeds. We do regular evaluations over several evaluation episodes throughout training, and their average return/success rate is what we report here. Configurations used for OFENet, including representation size and pretraining, are listed in Table 1 in the appendix. In Section A.4 in the appendix, we additionally examine and discuss the effect different representation sizes and amounts of pretraining would have on Hopper-v2, HalfCheetah-v2, and Humanoid-v2.

For our experiments we use the PyTorch implementations of TD3 by Fujimoto (2022) and SAC by Yarats & Kostrikov (2022), together with our own PyTorch implementation of OFENet based on the Tensorflow code provided by Ota et al. (2020). For the experiments with HER, we modified the Stable-Baselines3 code by Raffin et al. (2021) to include OFENet. Hyperparameters we used for all runs are reported in the appendix, Section A.2. For the experiments done with Stable-Baselines3, we took hyperparameters from the RL Baselines3 Zoo repository (Raffin (2020)). In all other experiments, hyperparameters are either the default ones provided by the respective RL algorithm implementation or the OFENet implementation of Ota et al. (2020).

In all cases, we pretrain OFENet by a certain number of steps which are reported in Section A.2 of the appendix. We set training steps to 0 when pretraining is completed and the RL algorithm starts training. For the remaining time, the system alternates between training OFENet on its auxiliary task and the RL algorithm on its RL task, while freezing the weights of the respective other. Representations are thus continuously updated during the training process and become optimized on those states and actions relevant to the agent.

In terms of computation time, adding OFENet to the RL algorithms roughly doubles to trebles the training time of our agents, which appears little given the stark increase in dimensionality. We speculate that this factor is mainly caused by a doubling in backward passes for gradient updates plus some additional overhead in handling two separate networks for two separate tasks.

6 RESULTS

The returns or success rates on all different environments are shown in Figures 4 and 5 for all auxiliary tasks and baseline algorithms. For a more normalized comparison, Figure 6 plots the normalized maximum return/success rate against sample efficiency. To measure sample efficiency, we calculate the fraction of training steps (and therefore samples) which are required to reach 80% of the maximum return of the baseline algorithm, calibrated against the untrained baseline algorithm. We call this measure SE80. Section A.3 of the appendix presents a discussion on the appropriate percentage value.

In the following, the word *performance* shall refer to the combination of maximum return and sample efficiency. If only one of the two is concerned, it will be named explicitly. It is apparent that all three auxiliary tasks lead to a significant increase in performance for high-dimensional, complex environments. For the low-dimensional and simple Pendulum-v1, a slight increase in sample efficiency but not in best return can be achieved. In fact, improvements in sample efficiency are achieved across almost all environments. Increases in maximum returns follow a certain pattern: they seem to increase with problem complexity rather than strictly dimensionality, although the two go hand in hand. When using HER to solve FetchSlideDense-v1, however, representation learning only leads to minor improvements. This is a special case discussed in Subsection 6.1.

6.1 REPRESENTATION LEARNING FOR DIFFERENT TYPES OF ENVIRONMENTS

Our experiments show different behavior for small and simple environments compared to larger and more complex environments. The small environments we study here can easily be solved by baseline TD3 or SAC and none of the auxiliary tasks leads to a noteworthy increase in maximum returns. For the very small and simple Pendulum-v1 environment, representation learning with auxiliary tasks does not significantly benefit sample efficiency either. For the slightly larger and less simple Hopper environment, the picture is ambiguous with an increase in sample efficiency for TD3 but not for SAC. For the remaining larger and more complex environments, however, representation learning with auxiliary tasks provides clear performance gains over baseline TD3 and SAC. These gains seem to

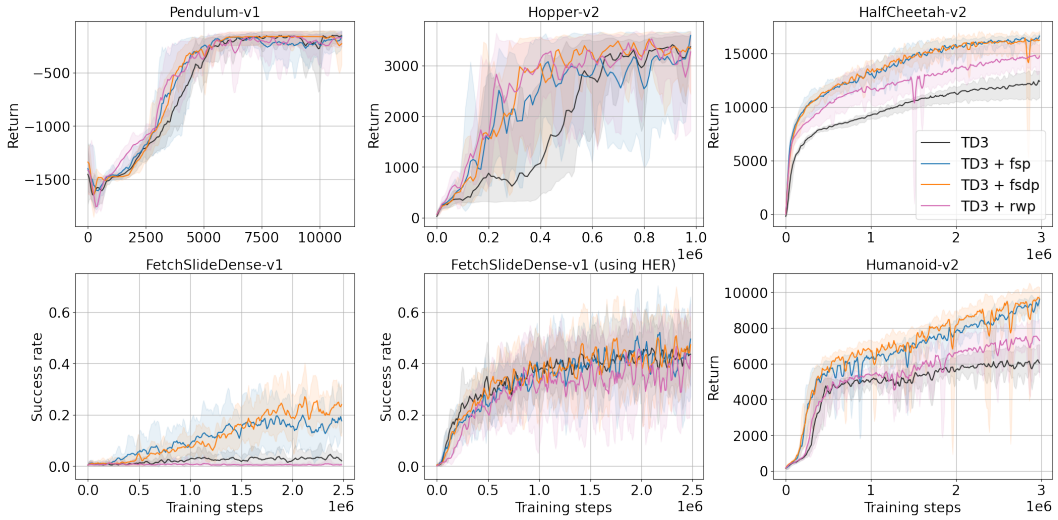

Figure 4: Returns/success rates achieved with TD3 and different auxiliary tasks on various environments. The shaded areas show minimum and maximum performance achieved across 5 runs, while the lines represent the means. Values have been smoothed slightly for better visualisation.

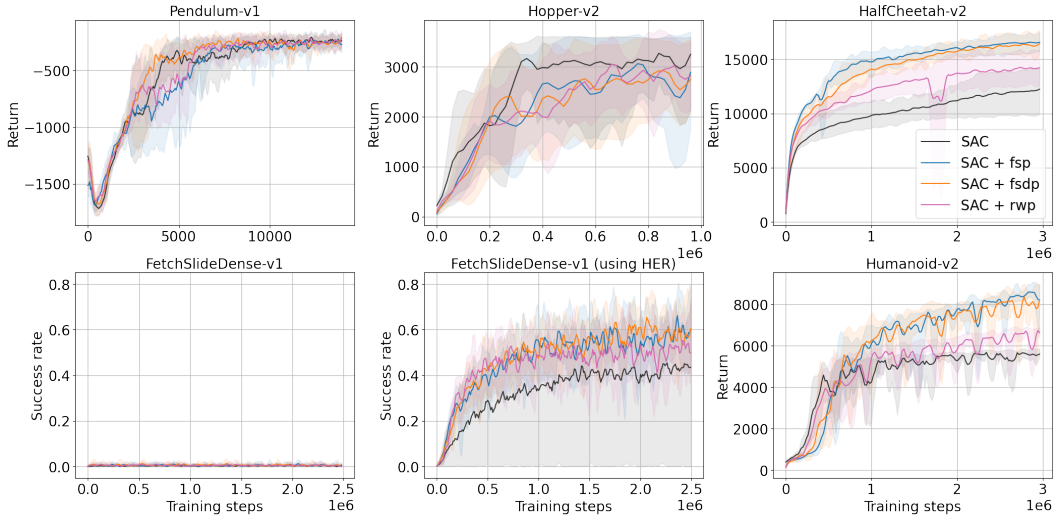

Figure 5: Returns/success rates achieved with SAC and different auxiliary tasks on various environments. The shaded areas show minimum and maximum performance achieved across 5 runs, while the lines represent the means. Values have been smoothed slightly for better visualisation.

scale with complexity rather than size of the environments, as the difference in performance between HalfCheetah-v2, FetchSlideDense-v1 and Humanoid-v2 is not proportionate to their difference in size.

An interesting case is the FetchSlideDense-v1 environment. It is too complex for any learning to occur with baseline TD3 or SAC (without HER). Because of its initially constant rewards, *rwp* is not able to learn anything at all. Adding HER to the RL algorithm, however, seems to speed up learning enough to generate meaningful rather than trivial reward signals very soon and to successfully train *rwp*, as evidenced by the fact that agents using *rwp* are competitive with those trained on other tasks.

The inventors of HER, Andrychowicz et al. (2017), argue that in cases such as FetchSlideDense-v1 too few learning impulses, in the form of rewards, are provided to meaningfully update network

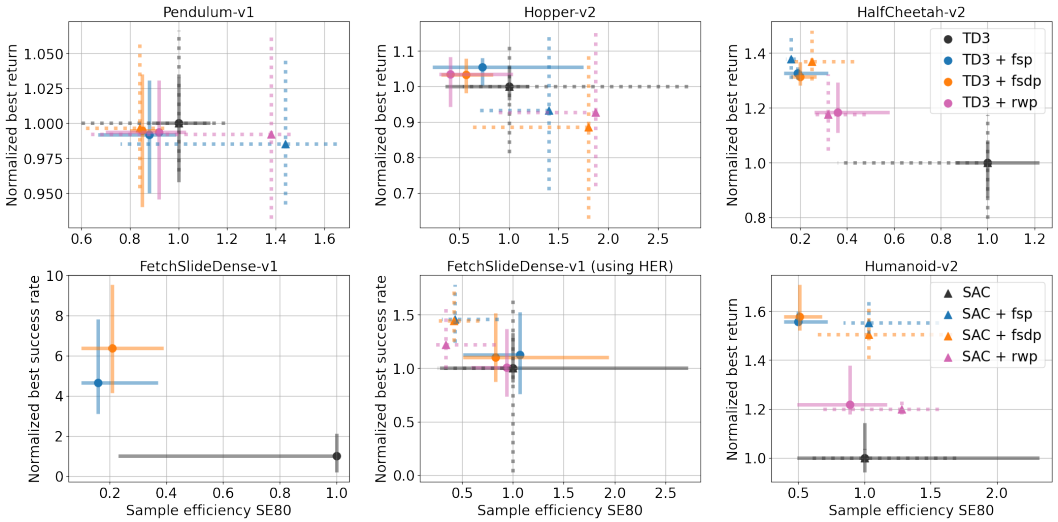

Figure 6: Sample efficiency on different environments compared to normalized best returns/success rates. Note the different scales on the axes. The markers describe average performance, error bars (solid for TD3 and dotted for SAC) mark best and worst case out of 5 runs. Where markers or error bars are missing, agents in question never reached the return/success rate required to calculate SE80.

weights in the RL algorithm. When using TD3 with HER, the auxiliary tasks do not seem to offer any benefits. For SAC with HER, the agents trained with auxiliary tasks are on average better than those without. This is misleading as baseline SAC with HER can perform as well as with auxiliary tasks, but its mean is lowered considerably by two agents which did not learn at all. These results suggest that adding a learning signal through HER already enables the RL algorithm itself to learn meaningful patterns from original observations (i.e. HER significantly reduces the complexity of the problem), even if not quite as reliably as with additional representations learned on auxiliary tasks.

Furthermore, FetchSlideDense-v1 becomes at least partially solvable for TD3, even without HER, when *fsp* or *fsdp* are used. This is an interesting result which shows that even if an environment is too complex for a RL algorithm, adding representation learning might still enable successful training of agents. There is however no such improvement in the same experiment with SAC, which shows that this strategy has its limits. We hypothesize the learned representations recast observations, actions and thereby the entire RL problem into a less complex manifold. At least some dimensions of the representation learned with OFENet would then contain more informative features than the original observation. For FetchSlideDense-v1 the representation might for instance contain a feature encoding distance between arm and puck, instead of just the absolute positions from raw observations. When the arm accidentally hits the puck, the RL algorithm could then relate observation and reward more easily. Another possible factor, proposed by Ota et al. (2020), is that the added depth and width of OFENet enable the agent to learn more complex and therefore more successful solutions. In this case, however, additional expressivity through added weights alone does not reduce problem complexity which is caused by initially constant rewards. It can therefore not explain why *fsp* and *fsdp* make FetchSlideDense-v1 learnable for TD3 without HER. We thus consider simplification of the learning problem to be the dominant factor at least in this environment.

6.2 Comparison of Auxiliary Tasks

This section presents a direct comparison of auxiliary tasks across the different algorithms and environments. Since HER seems to significantly distort the performance of auxiliary tasks compared to just using baseline RL algorithms, the FetchSlideDense-v1 solved with HER will not be considered.

The *rwp* task performs worst out of the three investigated auxiliary tasks. For the complex and high-dimensional environments it is quickly overtaken by *fsp* and *fsdp*, even though it appears competitive for environments with less complex dynamics where the choice of auxiliary task is not very

important. The superior performances of *fsp* and *fsdp* are approximately on par, although one of the two usually outperforms the other by a slight but noteworthy margin. There is however no apparent pattern to this. When used with TD3, there might be a slight tendency for *fsdp* to outperform *fsp*, but results are too inconclusive to confidently make this claim, especially since it cannot be observed with SAC-based agents.

There are two obvious causal factors which might explain why *rwp* performs worse. Firstly, due to its dimensionality alone, the prediction target r_{t+1} of *rwp* does not convey the same amount of information as the prediction targets of *fsp* and *fsdp*, which might slow down *rwp* compared to the other two. Secondly, representations learned on environment dynamics will likely contain environment information that is not as easily accessible by only predicting rewards with the original RL task or *rwp* auxiliary task. The latter point would underline the redundancy claim regarding *rwp*. However, neither of the two factors is easy to investigate without studying the representations. Their large dimensionality makes such an endeavor difficult and we defer this to future work.

The absence of a consistent difference in performance between *fsp* and *fsdp* suggests that the theoretical advantages of each, discussed in Section 3, are either not very important or cancel each other out. Our studied environments are well behaved since they all simulate real world physics. Consequently, they do not confront the algorithm with abrupt state changes or excessive noise. The fact that *fsp* on average still works about as well as *fsdp*, despite those properties, suggests that the advantages considered for *fsdp* in particular do not play a large role in practice.

7 Conclusion

In this paper we have compared common auxiliary tasks for representation learning in RL. To this end we have used five common continuous control benchmark environments and two different state-of-the-art off-policy RL algorithms. The representation learning is done explicitly, i.e. decoupled from the RL algorithm with a distinct neural network, rather than implicitly. Representations are then used as input to the RL algorithm instead of observations or, in case of the critic, actions.

We find that representation learning in general significantly improves both sample efficiency and maximum returns for larger and more complex environments. It makes little difference with smaller and simpler environments, where we observe a slight increase in sample efficiency at most. In general, auxiliary tasks which encourage learning environment dynamics considerably outperform reward prediction. A particularly encouraging result is that the FetchSlideDense-v1 environment, a simulated robotics arm, becomes partially solvable when adding representation learning to the otherwise unsuccessful TD3 algorithm. We interpret this as an indication that explicit representation learning with auxiliary tasks can reduce problem complexity in RL. Across all experiments, we found that results were quite different with different RL algorithms, even when using the same representation learning techniques.

Due to the prohibitive computational costs of training hundreds of agents, we were limited in the amount of environments and auxiliary tasks we could investigate. In the future, however, it would be interesting to compare tasks also on non-physical, stochastic and noisy environments. In addition, our comparison might be extended with further, less common auxiliary tasks or combinations of multiple tasks as described in Section 3.

REFERENCES

Charles W. Anderson, Minwoo Lee, and Daniel L. Elliott. Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, July 2015. doi: 10.1109/IJCNN.2015.7280824. ISSN: 2161-4407.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016. URL http://arxiv.org/abs/1606.01540. arXiv:1606.01540 [cs].

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuska. Integrating State Representation Learning Into Deep Reinforcement Learning. *IEEE Robot. Autom. Lett.*, 3(3):1394–1401, July 2018. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2018.2800101. URL http://ieeexplore.ieee.org/document/8276247/.

Scott Fujimoto. Addressing Function Approximation Error in Actor-Critic Methods, September 2022. URL https://github.com/sfujim/TD3. original-date: 2018-02-22T18:15:37Z.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods, October 2018. URL http://arxiv.org/abs/1802.09477. arXiv:1802.09477 [cs, stat].

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, August 2018. URL http://arxiv.org/abs/1801.01290. arXiv:1801.01290 [cs, stat].

Hlynur Davíð Hlynsson and Laurenz Wiskott. Reward prediction for representation learning and reward shaping, May 2021. URL http://arxiv.org/abs/2105.03172. arXiv:2105.03172 [cs, stat].

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks, November 2016. URL http://arxiv.org/abs/1611.05397. arXiv:1611.05397 [cs].

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Auton Robot*, 39(3):407–428, October 2015. ISSN 0929-5593, 1573-7527. doi: 10.1007/s10514-015-9459-7. URL http://link.springer.com/10.1007/s10514-015-9459-7.

Robert Legenstein, Niko Wilbert, and Laurenz Wiskott. Reinforcement Learning on Slow Features of High-Dimensional Input Streams. *PLOS Computational Biology*, 6(8):e1000894, August 2010. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1000894. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000894. Publisher: Public Library of Science.

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-François Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, December 2018. ISSN 0893-6080. doi: 10.1016/j.neunet.2018.07.006. URL https://www.sciencedirect.com/science/article/pii/S0893608018302053.

Xingyu Lin, Harjatin Baweja, George Kantor, and David Held. Adaptive Auxiliary Task Weighting for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0e900ad84f63618452210ab8baae0218-Abstract.html.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. URL http://arxiv.org/abs/1312.5602. arXiv:1312.5602 [cs].

Jelle Munk, Jens Kober, and Robert Babuska. Learning state representation for deep actor-critic control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4667–4673, Las Vegas, NV, USA, December 2016. IEEE. ISBN 978-1-5090-1837-6. doi: 10.1109/CDC.2016.7798980. URL http://ieeexplore.ieee.org/document/7798980/.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value Prediction Network. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/ffbd6cbb019a1413183c8d08f2929307-Abstract.html.

Kei Ota, Tomoaki Oiki, Devesh Jha, Toshisada Mariyama, and Daniel Nikovski. Can Increasing Input Dimensionality Improve Deep Reinforcement Learning? In *International Conference on Machine Learning*, pp. 7424–7433. PMLR, November 2020. URL http://proceedings.mlr.press/v119/ota20a.html. ISSN: 2640-3498.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2778–2787. PMLR, July 2017. URL https://proceedings.mlr.press/v70/pathak17a.html. ISSN: 2640-3498.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research, March 2018. URL http://arxiv.org/abs/1802.09464. arXiv:1802.09464 [cs].

Antonin Raffin. RL Baselines3 Zoo, 2020. URL https://github.com/DLR-RM/rl-baselines3-zoo.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. ISSN 1533-7928. URL http://jmlr.org/papers/v22/20-1364.html.

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own Reward: Self-Supervision for Reinforcement Learning, March 2017. URL http://arxiv.org/abs/1612.07307. arXiv:1612.07307 [cs].

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling Representation Learning from Reinforcement Learning. *arXiv:2009.08319 [cs, stat]*, September 2020. URL http://arxiv.org/abs/2009.08319. arXiv: 2009.08319.

Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3928–3934, October 2016. doi: 10.1109/IROS.2016.7759578. ISSN: 2153-0866.

Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models, June 2015. URL http://arxiv.org/abs/1502.02251. arXiv:1502.02251 [cs, stat].

Denis Yarats and Ilya Kostrikov. Soft Actor-Critic (SAC) implementation in PyTorch, September 2022. URL https://github.com/denisyarats/pytorch_sac. original-date: 2020-01-22T15:51:24Z.

A APPENDIX

In Section A.1 of this appendix to discuss in detail why the inverse dynamics auxiliary task does not work with actor critic algorithms. In Section A.2 we describe hyperparameters for all our experiments to establish reproducibility of our results. Then, in Section A.3 we debate our SE80 measure of sample efficiency. Section A.4, finally, contains an investigation of the impact of parameters *representation size* and *amount of pretraining* for different auxiliary tasks on the three MuJoCo control environments.

A.1 INVERSE DYNAMICS WITH ACTOR-CRITIC ALGORITHMS

Representations z_{o_t} only depend on o_t (see Figure 2), while $z_{o_t,o_{t+1}}$ learned with the *inv* auxiliary task depend on both o_t and o_{t+1}. The z_{o_t} is passed into the actor and $z_{o_t,o_{t+1}}$ into the critic. The latter is a problem when updating the actor, which calculates the policy, during training. For policy updates, o_t is provided by the replay buffer and a_t then proposed by the actor. The policy is updated by backpropagating the gradient of the critic loss with respect to the proposed action. The critic loss, however, also depends on o_{t+1} which is dynamically retrieved from the environment for the proposed a_t. Backpropagation is thus not possible if the environment is not differentiable, which is usually the case. This is visualized in Figure 7. It is possible to additionally pass a_t into the critic to establish a direct differentiable connection between critic and actor. The gradient computed purely through this connection is naturally not the true gradient, as long as the loss still also depends on o_{t+1} which, in turn, depends on a_t. However, one might assume that in practice the contribution to the gradient through o_{t+1} amounts only to a minor perturbation and could be ignored when a_t is also passed directly into the critic. In our experiments we could empirically establish that this is not the case. Having any dependency of the critic loss on o_{t+1} whatsoever prevented learning completely.

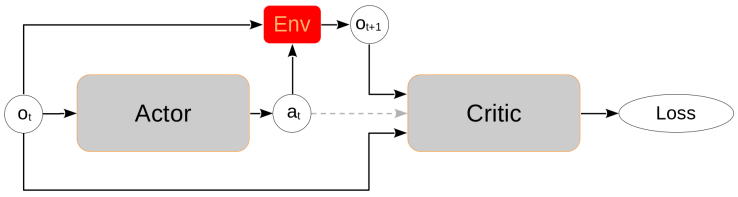

Figure 7: Diagram of information flow in an actor-critic setup with the *inv* auxiliary task where the critic receives o_t and o_{t+1} (and potentially a_t) as input. If o_{t+1} is part of the input to the critic, either directly or through a representation learning network such as OFENet, the gradient of the critic loss cannot be propagated back to the actor, as long as the environment (red) is not differentiable. This is true even if the action is additionally passed into the critic directly (dashed grey line).

A.2 EXPERIMENT PARAMETERS

For all environments we investigate, Table 1 lists the dimensionalities of observations o_t, actions a_t, of OFENet representations z_{o_t} and z_{o_t,a_t}, and finally also the OFENet configurations used to obtain them.

The hyperparameters we used for different runs were left largely untouched from their defaults in the respective implementation. Batch sizes and learning rates are reported in Table 2. For those runs without HER we have used defaults of the individual TD3 and SAC implementations, and configured OFENet according to the last column of Table 1. For the runs with HER, we have used defaults of Stable-Baselines3 but adjusted the ones suggested in RL Baselines3 Zoo for FetchSlide-v1 according to values proposed in this file.

OFENet was pretrained with 10000 steps for all environments except Pendulum-v2, where 1000 pretraining steps were used. In Section A.4 of this appendix we study how a change in pretraining steps between values 0, 10000 and 100000 affects results for Hopper-v2, HalfCheetah-v2, and Humanoid-v2.

Table 1: Dimensions of observations, actions, representations and OFENet parameters used to achieve them. Layers per part describes the total amount, and individual width, of fully connected layers per OFENet part. Numbers in brackets are lower and higher representation sizes used in the comparison in Section A.4

Environment	dim(o_t)	dim(a_t)	dim(z_{o_t})	dim(z_{o_t,a_t})	Layers/part
Pendulum-v1	3	1	23	44	2 x 10
Hopper-v2	11	3	251 (59, 971)	494 (110, 1934)	6 x 40 (x8, x160)
HalfCheetah-v2	17	6	257 (65, 977)	503 (119, 1943)	8 x 30 (x6, x120)
FetchSlideDense-v1	31	4	271	515	8 x 30
Humanoid-v2	292	17	532 (340, 1252)	789 (405, 2229)	8 x 30 (x6, x120)

Table 2: Batch sizes and learning rates used for training different agents.

Agent	Batch size	Learning rate
SAC	256	0.0003
SAC + HER	2048	0.001
TD3	256	0.0003
TD3 + HER	2048	0.001

A.3 Measuring sample efficiency

In the literature, sample efficiency is commonly measured by setting a return threshold and evaluating how many training steps are required to reach it. In our case, we want to compare agents relative to some baseline agent. We therefore make the threshold dependent on maximum performance of the baseline agent. This raises the question what percentage of that maximum performance is a reasonable pick for a threshold. We select 80% as a good threshold for our comparison. It is high enough to yield sample efficiency values similar to a threshold of 100%, i.e. the 80% values reasonably reflect how much faster an alternative can reach full maximum performance of the baseline algorithm. On the other hand, 80% can also reflect how agents with auxiliary tasks (or, in some cases, the baseline algorithms) can learn a bit faster before reaching the same value of maximum performance. When all agents reach approximately the same maximum performance, as in Pendulum-v1 and Hopper-v2, setting the threshold to 100% is also problematic for a second reason: Sample efficiency can not be calculated at all, as soon as maximum performance of the baseline algorithm happens to be even just a tiny fraction larger than the maximum performance of agents with auxiliary tasks.

Figures 8 and 9 plot how the sample efficiency measure (vertical axis) changes with different percent values for the threshold (horizontal axis). Even though we initially generated these figures merely to establish a reasonable percentage value, it turned out that effects visible in these plots are quite interesting. They make explicit how agents using auxiliary tasks, on some environments, learn much quicker than the baseline algorithms right from the start of training (e.g. HalfCheetah-v2 or FetchSlideDense-v1 with TD3). On other environments, however, the advantage only builds over time (HalfCheetah-v2 or FetchSlideDense-v1 with HER on SAC). On Humanoid-v2, finally, an increase in sample efficiency only occurs once the baseline algorithm approaches asymptotic behavior and is overtaken confidently by agents that use auxiliary tasks.

A.4 Effects of Representation Size and Pretraining

In addition to the direct comparison of auxiliary tasks, we also investigate the effect of hyperparameters *representation size* and *amount of pretraining* on performance. Figures 10 and 11 show how average performances change for different auxiliary tasks when these are modified. The markers of intermediate size (pretraining of 10000 and representations with 240 added dimensions) correspond to the experiments discussed in Section 6, while smaller and larger markers represent increased and decreased values for the respective hyperparameter. We have only conducted these experiments for

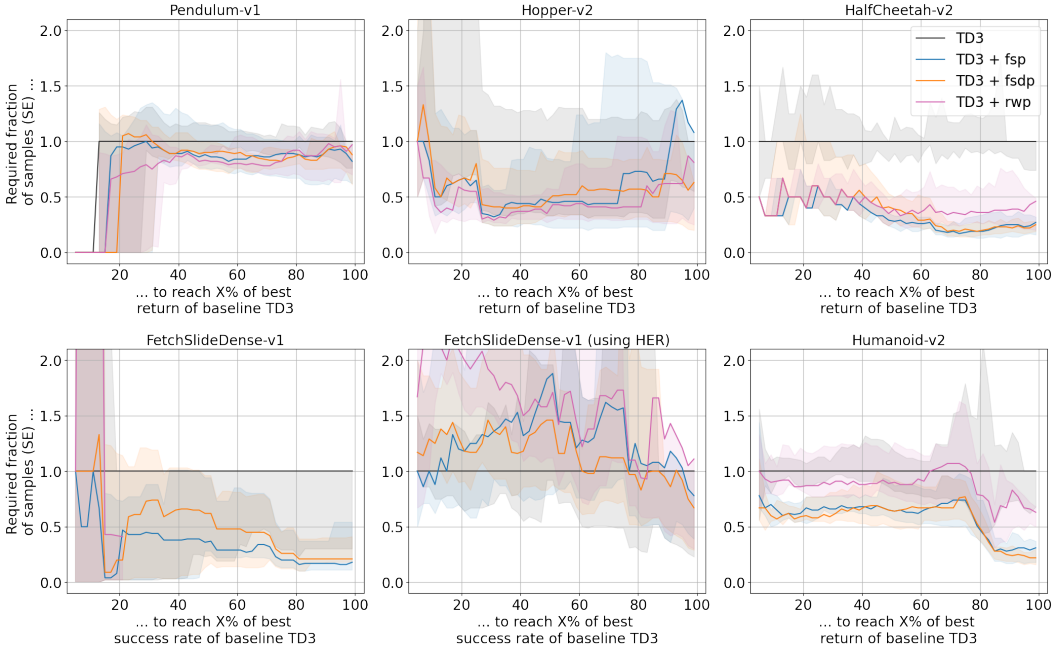

Figure 8: Sample efficiencies of auxiliary tasks used with TD3, for different environments. If values are missing – either of the mean or the min-max range – this is because their return/success rate never reached the baseline return/success rate.

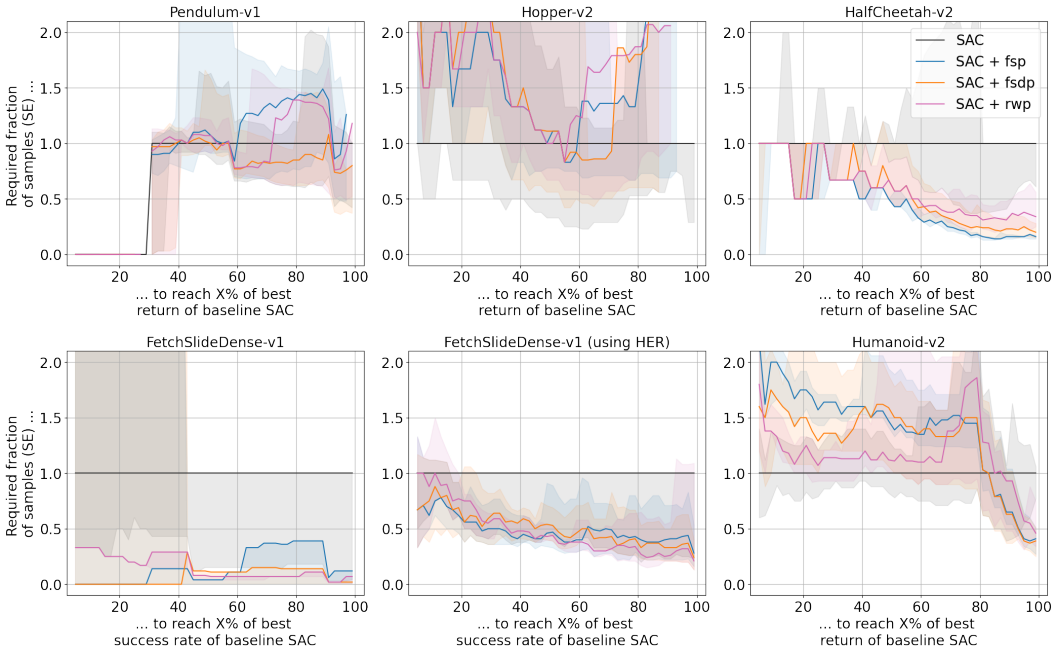

Figure 9: Sample efficiencies of auxiliary tasks used with SAC, for different environments. If values are missing – either of the mean or the min-max range – this is because their return/success rate never reached the baseline return/success rate. Note that the plot for FetchSlideDense-v1 without HER is misleading as it is based on agents that have not learned anything and behave randomly (see Figure 5).

Hopper-v2, HalfCheetah-v2 and Humanoid-v2 since such an investigation becomes very expensive due to the amount of different configurations.

While the results are certainly interesting, our findings exhibit limited consistency across different hyperparameter values and do not directly affect our main comparison of auxiliary tasks in Section 6.

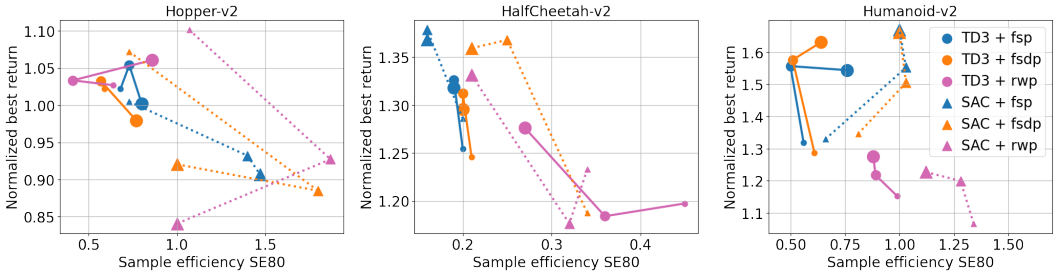

Figure 10: Comparison of different representation sizes. Within OFENet 48, 240 or 960 dimensions were added to the original observations and actions (see Table 1). The marker size corresponds to the representation size. Each marker represents a mean across five runs with different seeds.

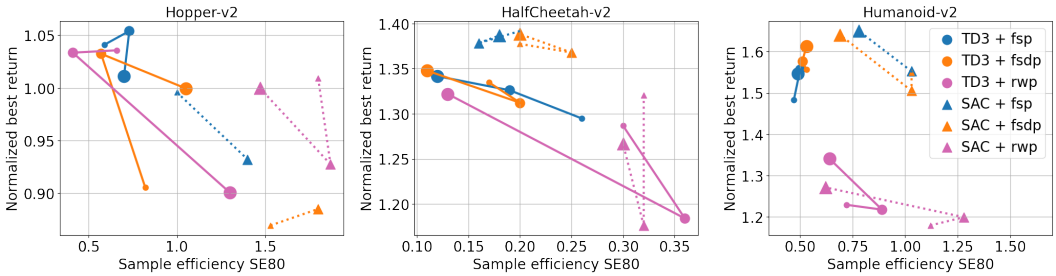

Figure 11: Comparison of different amounts of pretraining. The pretraining was performed for 0, 10000 or 100000 steps. The marker size corresponds to the amount of pretraining. Each marker represents a mean across five runs with different seeds.

For representation sizes used with SAC and *fsp* or *fsdp*, it is apparent that smaller environments can be solved better with smaller representations and larger environments with larger representations. For TD3, representations of medium size generally lead to the best trade-off between maximum return and sample efficiency. In general, differences in representation sizes seem to cause smaller performance changes than with SAC. For both TD3 and SAC there is a significant amount of noise to the respective pattern.

The picture is somewhat different for varying amounts of pretraining. There are variations in performance but effects are somewhat inconsistent. For Hopper-v2, the only apparent result is that the largest amount of pretraining hurts performance in all cases except *rwp* with SAC. For HalfCheetah-v2, maximum returns are barely affected, but at least for TD3 a large amount of pretraining seems to improve sample efficiency. For SAC, there are no clear patterns to what amount of pretraining best benefits sample efficiency. Lastly, for Humanoid-v2, the largest amount of pretraining seems to correlate with the largest performance gains. With TD3, specifically, improvements are however mostly in maximum returns rather than sample efficiency.