Out-of-distribution generalization of internal models is correlated with reward

Khushdeep S. Mann[*] TU Berlin
Steffen Schneider[*†] EPFL & U. Tübingen
Alberto Chiappa EPFL
Jin H. Lee TU Munich
Matthias Bethge University Tübingen
Alexander Mathis EPFL
Mackenzie W. Mathis EPFL

Abstract

We investigate the behavior of reinforcement learning (RL) agents under morphological distribution shifts. Similar to recent robustness benchmarks in computer vision, we train algorithms on selected RL environments and test transfer performance on perturbed environments. We specifically test perturbations to popular RL agent's morphologies by changing the length and mass of limbs, which in biological settings is a major challenge (e.g., after injury or during growth). In this setup, called PyBullet-M, we compare the performance of policies obtained by reward-driven learning with self-supervised models of the observed state-action transitions. We find that out-of-distribution performance of self-supervised models is correlated to degradation in reward.

1 Introduction

Animal behavior is highly adaptive. When challenged with different terrain, as well as changes to their bodies (such as growth or injury), they operate successfully by adapting their behavior to new conditions. In comparison to animals, robots and artificial autonomous agents are less versatile in recovering task-effective behavior, when properties of their body, or of the surrounding environment change. This discrepancy can be attributed to the lack of an internal model of the world, which has been demonstrated in humans and other animals (Wolpert et al., 1995; Kawato & Wolpert, 1998).

In reinforcement learning (RL) robustness and generalization are two main concerns, especially in robotics applications, where a policy trained in simulation should adapt to the different conditions in the real world. However, standard RL algorithms are designed to learn a policy which yields the maximum reward given the environment's transition dynamics. As analyzed by Koos et al. (2010) and Peng et al. (2018), this is one of the main causes of the reality-gap problem.

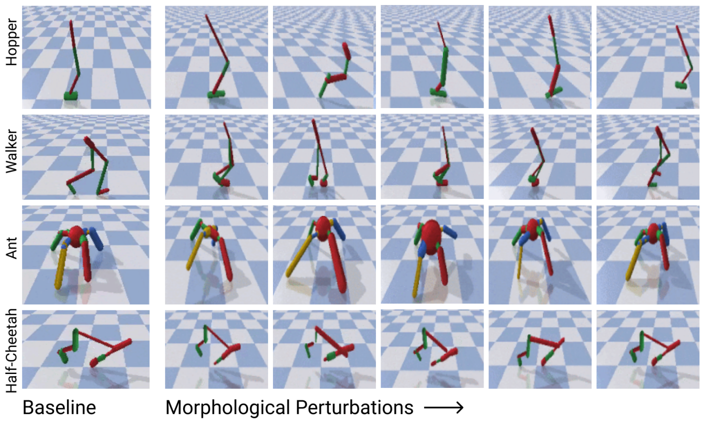


Figure 1: Benchmarking robustness to morphological perturbations in PyBullet. The modified morphology of the 4 agents is a challenging test for the policies trained on baseline.

Inspired by recent work using environmental perturbations (Packer et al., 2019; Pathak et al., 2019; Mankowitz et al., 2019), we study how the reward in motor control tasks decays when the morphology of the agent changes. In particular, we randomize length and size of limbs

[*]Equal contribution; † correspondence: steffen@bethgelab.org

at test time (without additional learning) and measure the loss in performance as a function of the modification's severity. Furthermore, we test the utility of a self-supervision task as a metric for generalization success. Our experiments show a correlation between the task performance and the accuracy in solving a self-supervised task, which motivates the choice of self-supervision for future work as a way to recover task performance at test time without providing reward feedback.

Related Work. Robustness and generalization are key challenges in machine learning. In computer vision benchmarks have driven rapid algorithmic innovations, and in particular recent benchmarks such as ImageNet-C (Hendrycks & Dietterich, 2019), that apply common corrupts to images have both revealed limitations, and allowed the development of new models that are more robust in real-world scenarios (e.g., Xie et al., 2020; Hendrycks et al., 2020; Geirhos et al., 2020). In RL, the *de facto* standard benchmarks test the performance of algorithms across many tasks, e.g. in Arcade (Bellemare et al., 2013; Mnih et al., 2013) and rllab (Duan et al., 2016). Recently, new benchmarks for generalization in continuous control tasks have been put forth (Packer et al., 2019; Dulac-Arnold et al., 2021).

Several techniques have been proposed for RL algorithms to handle test environments, where the agent faces perturbed transition dynamics compared to what it had been trained on. Meta-learning techniques have proven successful for domain adaptation (e.g. Nagabandi et al., 2018). For improving the robustness of RL algorithms, strategies such as data augmentation (Kostrikov et al., 2020; Laskin et al., 2020), representation learning (Srinivas et al., 2020; Raffin & Stulp, 2020), self-supervision (Hansen et al., 2020; Sun et al., 2019) and worst-case reward optimization (Mankowitz et al., 2019) have shown promising results. In these studies the agent is either tested in novel environments or the agent's physical properties (morphology, mechanics) are modified. Pathak et al. (2019) even allow the agent structure to change completely through the recombination of body parts.

2 Methods

We consider continuous control problems implemented in the PyBullet library (Coumans & Bai, 2016–2019) and use the environments Hopper, Walker, Half-Cheetah and Ant (Figure 1). We evaluate the performance of two learning schemes: Reward-driven learning for policies and contrastive self-supervised learning for modeling the dynamics.

Reward-driven learning. We train two state of the art model-free, off-policy actor-critic algorithms, soft-actor critic (SAC; Haarnoja et al., 2018) and twin-delayed deep deterministic policy gradient (TD3; Fujimoto et al., 2018). The objective of the RL agent is to maximize a reward signal proportional to the (signed) linear velocity in the forward direction. The state $\boldsymbol{s}$ of the system is fully observable, as the observations include the relative position of each body component, as well as the velocity, inertia, internal and external forces. The policy acts by selecting the torque to apply at each joint per time step. TD3 maximizes the reward by updating the policy with deterministic policy gradient (Silver et al., 2014) and enabling the training of deep networks by including a second critic network, a target network and other techniques such as batch normalization. In SAC an additional entropy term is added (modulated by a parameter α), which controls the stochasticity of the policy (Haarnoja et al., 2018). Additional details are given in §A.1.

Self-supervised learning. We use a time contrastive loss (Hyvärinen & Morioka, 2016; Oord et al., 2018) for self-supervised learning (SSL) and adapt the model setup of Schneider et al. (2019) to use three convolutional networks. The state encoder f_s maps single states $\boldsymbol{s}$ and the action encoder f_a maps single actions $\boldsymbol{a}$ to features. The aggregator network g computes a *context representation* from the past t' state-action pairs,

$$\boldsymbol{c}_t = g(f_s(\boldsymbol{s}_{t-1}), \ldots, f_s(\boldsymbol{s}_{t-t'}), f_a(\boldsymbol{a}_{t-1}), \ldots, f_a(\boldsymbol{a}_{t-t'})), \tag{1}$$

and we consider a variant where only states $\boldsymbol{s}$ are considered. We train the networks to minimize the contrastive learning criterion,

$$\mathcal{L}_k = -\sum_{i=1}^{T-k} \Big(\log \sigma(f_s(\boldsymbol{s}_{i+k})^\top h_k(\boldsymbol{c}_i)) + \lambda \mathbb{E}_{\tilde{\boldsymbol{s}} \sim p_n} \left[\log \sigma(-f_s(\tilde{\boldsymbol{s}})^\top h_k(\boldsymbol{c}_i)) \right] \Big), \tag{2}$$

for multiple time steps k, where $\sigma(\cdot)$ is the sigmoid function and $\tilde{s}$ is a random negative sample from the state sequence. Intuitively, this is a binary classification task between a "true" state $\boldsymbol{s}_{i+k}$

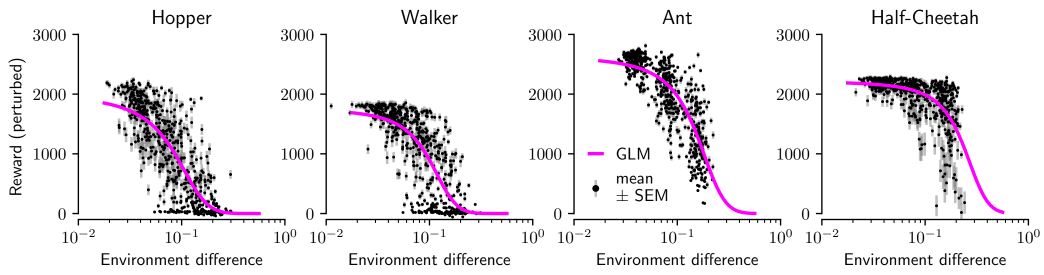


Figure 2: The difference between baseline and perturbed morphology (L^1 difference in lengths and widths of limbs) is correlated with reward in perturbed environment for TD3 (mean across seeds). We show results for SAC in Figure 6.

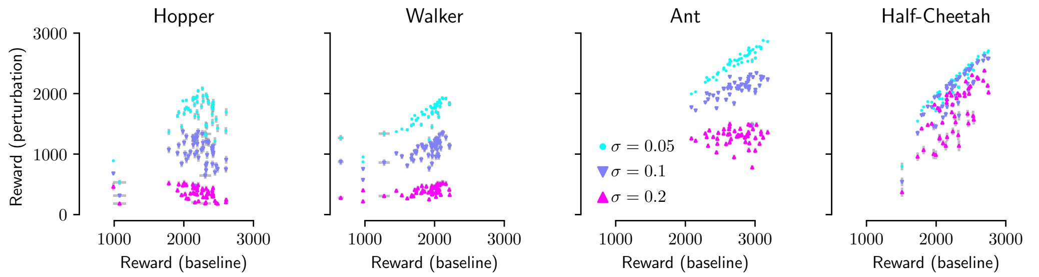


Figure 3: Comparing baseline to target reward reveals differences between agents: Baseline reward is indicative of robustness with best correlation for Ant and Half-Cheetah for TD3 (mean across envs). We show results for SAC in Figure 7.

following k steps after the context $\boldsymbol{c}_t$ and a set of distractor samples $\tilde{\boldsymbol{s}}$. As an intuitive metric, we can compute the true positive rate as the fraction of positive samples $\boldsymbol{s}_{i+k}$ with $f_s(\boldsymbol{s}_{i+k})^\top h_k(\boldsymbol{c}_i) > 0$. See §A.2 for further details on the model, and §A.3 for details on the SSL data generation.

3 Experimental Protocol

Training in the baseline environments. We benchmark the behavior of TD3 (Fujimoto et al., 2018) (Lillicrap et al., 2015) and SAC (Haarnoja et al., 2018), using a two-layer MLP with hidden layer sizes 300 and 400 and ReLU activation functions. We outline the remaining (standard) hyperparameters in §A.1. We train 50 seeds for TD3. For SAC, we additionally investigate how the entropy parameter α (cf. §A.1) affects model robustness. We train 10 seeds for each value of $\alpha \in \{0.01, 0.033, 0.05, 0.1, 0.2, 0.33\}$. Training is done for 10^6 time steps in episodes with 10^3 steps maximum duration.

Testing in perturbed environments (PyBullet-M) In each of the considered environments, we vary the limb length and width for each agent. We sample from a normal distribution $\mathcal{N}(x_0, \sigma x_0)$ centered around the original limb width or length x_0 with a variance relative to the baseline quantity (Fig. 1). We obtain environments of varying difficulty by choosing $\sigma \in \{0.05, 0.1, 0.2\}$. We sample 200 random environments for each value σ. We report the median reward across 10^4 evaluation steps and we reset the environment after a maximum episode length of 10^3 steps. We repeat this evaluation 10 times for each baseline environment to collect additional data for SSL training.

Analysis. We compute the mean reward and standard error of the mean (SEM) across model seeds, yielding 200 data points per algorithm and agent. For relating environment difference and target reward (Fig. 2), we fit a Binomial GLM with logit link function to the reward ratio $r/(r_{\text{max}} - r)$. As a model for the relationship between positive sample accuracy in contrastive learning and the drop in reward $r_{\text{base}} - r$ (Fig. 4), we fit a Gaussian GLM with logarithmic link function. We fit GLMs with the statsmodels package (Seabold & Perktold, 2010).

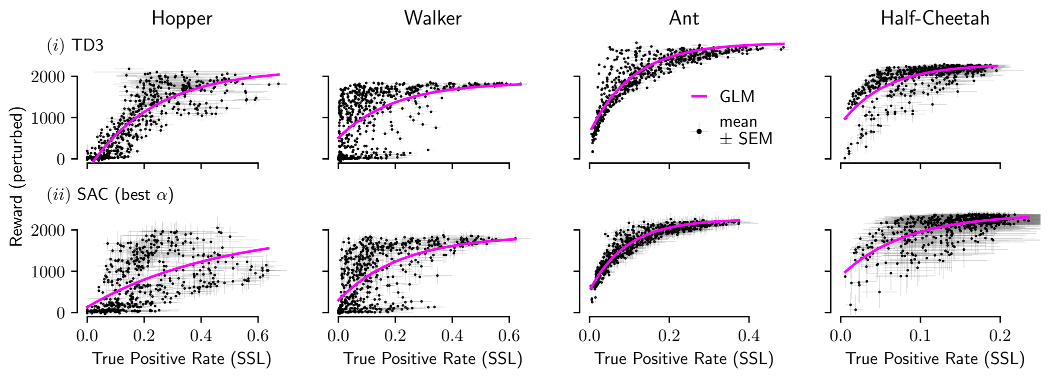


Figure 4: Out of distribution evaluation of models trained on the unperturbed environment. Policies were trained on non-perturbed environments using TD3 or SAC. SSL models were then trained on state-action pairs of these baseline policies. We show the results of testing both model types on the perturbed environments. The true positive rate of SSL models is correlated with the reward obtained by RL models (mean across seeds) during evaluation on the perturbed environments.

4 RESULTS

We focus our analysis on metrics with relationships to the reward obtained on the perturbed environments. The general approach is inspired by robustness benchmarks in computer vision: On datasets like ImageNet-C (Hendrycks & Dietterich, 2019), it is known that (in many cases) baseline accuracy correlates with target accuracy (Hendrycks et al., 2020). Moreover, we sought to find metrics (based on the baseline behavior) allowing to predict degradation in performance from the level of distribution shift (Schneider et al., 2020).

Environment mismatch correlates with degradation in reward. We first study the general properties of the PyBullet-M tasks. We compare the relative environment difference between each perturbed environment and the baseline setting for the respective agent to the reward obtained when testing on this environment. Degradation in reward correlates with differences in the environment, which we compute as the L^1 relative difference between baseline and perturbed limb configuration. We find that the relationship can be modeled by a logistic GLM (Fig. 2, 6). While we observe the full range of the logistic curve for Hopper and Walker environments, the Ant and Half-Cheetah environments are more robust, potentially because their body plan is inherently more stable. Going forward, we therefore contrast these "stable" agents (Ant, Half-Cheetah) to the metrics we observe on the unstable agents (Hopper, Walker).

Baseline correlates with out-of-distribution reward for stable environments & small perturbations. We now compare the performance obtained in the baseline and target environments (Fig. 3, 7). In low-perturbation environments ($\sigma = 0.05$), we see a strong correlation for the stable environments (Ant, Half-Cheetah). The relationship is generally less clear for the unstable environments (Walker, Hopper). On high-perturbation environments ($\sigma = 0.2$), the relationship only holds for Half-Cheetah. This result generally suggests that (for low perturbation environments) model selection for PyBullet-M is possible by using the baseline reward as a metric. On the other hand, given that the correlation weakens as perturbation strength increases, additional (unsupervised) model selection metrics are needed (without taking into account the algorithm performance on the target environment). We reasoned that self-supervised models trained on the dynamics of state-action pairs during baseline compared to perturbations could be a metric.

Out-of-distribution error of self-supervised learning predicts drop in reward. We train self-supervised models on all baseline environments (jointly for SAC and TD3 algorithms, 1100 runs total per agent, 11M total time steps) and compute their *ood.* performance when evaluated on perturbed environments. We use the true positive rate of the SSL models as an intuitive measure of performance, indicating how well the model is able to predict subsequent states up to $k = 16$ steps into the future from state-action pairs observed in the past. We find a log-linear relationship between the true positive rate (median across the $k = 16$ time steps) of the self-supervised model and the reward (Fig. 4, 9). The relationship holds particularly well for Ant, Hopper and Half-Cheetah environments. Overall this suggests that the out of distribution performance of "internal models" (trained on state-action pairs) can indeed serve as a metric for predicting the reward obtained by the baseline policies.

Additional results (Appendix, §B). We show baseline results for TD3 and SAC in §B.1, compare baseline vs. models *trained* on the perturbed environments in §B.2 and provide detailed results for the three aforementioned sections in §B.3–B.5, including an ablation for a purely state-based SSL model.

5 Conclusions

We analyze the performance of two state of the art actor-critic methods—SAC and TD3—under morphological changes of the agent body (in our new task suite, called PyBullet-M). We show correlations between *ood.* reward and differences in the environment. Under small environment changes, *ood.* reward is also well correlated with baseline reward. Contrasting the *ood.* performance of an "internal model" trained with self-supervised learning, correlates well with the expected reward obtained by reward-driven reinforcement learning. Taken together and consistent with other recent results (Hansen et al., 2020; Sun et al., 2019), we expect that exploring a causal link through self-supervised adaptation mechanisms in RL is an interesting research direction.

Author Contributions

StS conceived of the project; StS devised the experimental protocol with help of KM; KM implemented and ran the experiments; JHL conducted experiments and analysis on self-supervised training; AC, KM, and StS analyzed the data; StS, AC, MWM, AM, MB wrote the paper.

Acknowledgments

Funding was provided from the Bertarelli Foundation (StS, MWM), Swiss National Science Foundation (StS, MWM), the German Academic Exchange Service, DAAD (JHL), the International Max Planck Research School for Intelligent Systems (StS) and by the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center, FKZ: 01IS18039A (StS, JHL).

References

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pp. 1329–1338. PMLR, 2016.

Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning, 2021.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018.

Nicklas Hansen, Yu Sun, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. *arXiv preprint arXiv:2006.16241*, 2020.

Aapo Hyvärinen and Hiroshi Morioka. Unsupervised feature extraction by time-contrastive learning and nonlinear ica. In *NIPS*, 2016.

M. Kawato and D. Wolpert. Internal models for motor control. *Novartis Foundation symposium*, 218:291–304; discussion 304–7, 1998.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014. Citeseer, 2000.

Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. pp. 119–126, 2010. ISBN 9781450300728. URL `https://doi.org/10.1145/1830483.1830505`.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Daniel J Mankowitz, Nir Levine, Rae Jeong, Yuanyuan Shi, Jackie Kay, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy Mann, Todd Hester, and Martin Riedmiller. Robust reinforcement learning for continuous control with model misspecification. *arXiv preprint arXiv:1906.07516*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning, 2019.

Deepak Pathak, Chris Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. *arXiv preprint arXiv:1902.05546*, 2019.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics. *arXiv preprint arXiv:2005.05719*, 2020.

Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Removing covariate shift improves robustness against common corruptions. *Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. PMLR, 2014.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019.

Daniel M Wolpert, Zoubin Ghahramani, and Michael I Jordan. An internal model for sensorimotor integration. *Science*, 269(5232):1880–1882, 1995.

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

A Additional Experimental Details

A.1 Detailed Description of RL Agents.

All the results presented in this work have been obtained with two Reinforcement Learning algorithms: Twin Delayed Deep Deterministic Policy Gradient (TD3; Fujimoto et al., 2018) and Soft Actor Critic (SAC; Haarnoja et al., 2018). All of them belong to the family of the *policy gradient* algorithms, which allows them to deal with continuous action spaces, and they are thus ideally suited for the continuous motor control tasks presented in this paper. In particular, all of them are *actor-critic* algorithms (Konda & Tsitsiklis, 2000), meaning that they both train a policy function (actor) and an action-value function (critic).

The update rule of policy gradient algorithms is based on the gradient of an objective function, which usually is the expected return at the initial state:

$$J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[r\left(s,a\right)\right] \tag{3}$$

where $r(s,a)$ is the reward obtained by choosing action a at the state s, π_θ is the policy parametrized by θ and ρ^π the visitation distribution of the states under the policy π. This objective function is in fact used to derive, thanks to the deterministic policy gradient theorem (Silver et al., 2014), the policy update rule of DDPG (Lillicrap et al., 2015) and TD3:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi}\nabla_\theta\pi_\theta(s)\nabla_a Q^\pi(s,a)|_{a=\pi_\theta(s)} \tag{4}$$

where $Q^\pi(s,a)$ is the state-action value function (critic). The main difference between DDPG and TD3 is that the latter method addresses a Q-value overestimation problem of former thanks to the introduction of a second Q-network. On the other hand, SAC modifies the objective function by adding an auxiliary entropy term, which encourages the policy to maximize stochasticity. The objective writes as follows:

$$J(\pi) = \sum_{t=0}^{T}\mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right] \tag{5}$$

where α is a temperature parameter which regulates the weight of the entropy term $\mathcal{H}$. As we thought that there might be a relation between the stochasticity of the policy and its generalization performance, we ran tests for different values of α, as shown in § B.1.

Table 1: Hyperparameters for TD3 algorithm

Hyperparameters	Value
Discount factor γ	9.9×10^{-1}
Soft target network update parameter τ	5×10^{-3}
Batch size	256
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Training steps	10^6
Exploration steps	10^3
Maximum steps for each episode	10^3
Replay buffer size	10^5
Dimensions of first hidden layer for actor and critic	400 units
Dimensions of second hidden layer for actor and critic	300 units
Exploration noise	0.1
Policy noise	0.2
Noise clip	0.5
Nonlinearity	ReLU
Optimizer	Adam (Kingma & Ba, 2014)

Table 2: Hyperparameters for the SAC algorithm.

Hyperparameters	Value
Discount factor γ	9.9×10^{-1}
Soft target network update parameter τ	5×10^{-3}
Alpha α	$[0.01, 0.033, 0.05, 0.1, 0.2, 0.33]$
Batch size	256
Actor learning rate	3×10^{-4}
Critic learning rate	3×10^{-4}
Policy learning rate	3×10^{-4}
Training steps	10^6
Exploration steps	10^3
Maximum steps for each episode	10^3
Replay buffer size	10^5
Dimensions of first hidden layer for actor and critic	400 units
Dimensions of second hidden layer for actor and critic	300 units
Gradient steps (updates per step)	1
Target update interval	1
Automatic entropy tuning	False
Log sig max	2
Log sig min	-20
Epsilon ϵ	10^{-6}
Nonlinearity	ReLU
Optimizer	Adam(Kingma & Ba (2014))

A.2 Self Supervised Model Training

We train self-supervised models on state-action pairs obtained by the evaluation runs of the baseline policies. As a model for time-series, we adapt the wav2vec architecture (Schneider et al., 2019) originally designed for self-supervised learning (SSL) in speech processing. The state encoders f_s and the action encoder f_a are CNNs with kernel sizes (4, 4), strides (2, 2), dilation factors (2, 1) and channel sizes (64, 128). This yields a receptive field size of 16 and the output features are subsampled by a factor of four. The aggregator consists of two CNN layers with kernel sizes (3, 3), strides (1, 1), dilation factor (1, 1) and channel sizes (256, 128). The batch size is fixed to 10,000 tokens, the input token size per sample in each batch for training is set to 500 and 10 negative samples are sampled for the contrastive loss. We predict $k = 16$ tokens into the future and compute the true positive rate for each of these time steps. We train for 40k updates with the Adam optimizer (Kingma & Ba, 2014) and cosine learning rate schedule (Loshchilov & Hutter, 2016; Schneider et al., 2019). The learning rate is initialized to 10^{-6} and then gradually increases up to $5 \cdot 10^{-4}$ over 500 updates then decrease in cosine curve to 10^{-9}.

A.3 Training Data for Self Supervised Learning

We build one dataset per agent. We combine the state-action pairs obtained on the baseline environment. We use 50 model seeds for TD3 and 10 seeds per 6 alpha values for SAC (60 models). We obtain 1100 total runs with 11M tokens per agent, since each model is evaluated 10 times for 10^4 total evaluation steps. The total dataset size per agent contains approximately 11M tokens. The dataset is randomly split into a 9:1 training:validation ratio. After training, we pick the best model according to validation performance and evaluate the value of the contrastive loss as well as the true positive rate for each time step of the perturbed environments.

B Additional Results

We here supplement the three key results in the paper by additional baseline and control experiments, as well as more detailed versions of the paper plots. The section is organized as follows: Baseline results are outlined in §B.1. In §B.2, we test if some environments are inherently more difficult than others by computing topline results. We conclude with §B.3,B.4,B.5 which contains more detailed version of the plots used in the result section of the main text.

B.1 Baseline Results

We train 50 seeds for TD3 and 10 seeds of SAC for 6 different settings of α. The full baseline results are depicted in Table 3. For all summary plots in the paper, the best α values for SAC based on this baseline performance are reported.

Table 3: Baseline results (mean $\pm$ SEM across seeds), using standard hyperparameters for both SAC and TD3. We run an ablation across the regularizer value α in SAC to explore different behaviors based on the stochasticity of the policy (high α encourages high entropy).

Algorithm	α	Hopper	Walker	Ant	Half-Cheetah
SAC	0.010	2260 $\pm$ 39	1735 $\pm$ 15	1737 $\pm$ 96	2224 $\pm$ 25
	0.033	2040 $\pm$ 65	1801 $\pm$ 34	2023 $\pm$ 88	2233 $\pm$ 21
	0.050	2250 $\pm$ 49	1781 $\pm$ 8	2191 $\pm$ 46	2360 $\pm$ 25
	0.100	1949 $\pm$ 58	1506 $\pm$ 34	2054 $\pm$ 30	1945 $\pm$ 62
	0.200	2208 $\pm$ 15	916 $\pm$ 28	905 $\pm$ 10	808 $\pm$ 6
	0.330	1621 $\pm$ 38	704 $\pm$ 3	651 $\pm$ 5	751 $\pm$ 7
TD3	—	2186 $\pm$ 13	1816 $\pm$ 18	2723 $\pm$ 12	2253 $\pm$ 12

B.2 Control: Environment difficulty and topline performance.

For each of the 200 evaluation settings, we compute both the baseline and topline performance. Baseline performance is obtained by fitting the agent on the respective baseline environment, using standard hyperparameters, and subsequently evaluating on the perturbed environment. We also compute the topline performance, by training and evaluating agents on each of the perturbed environments, given an intuitive estimate of environment difficulty. Results for TD3 and SAC ($\alpha = 0.2$) are depicted in Fig. 5.

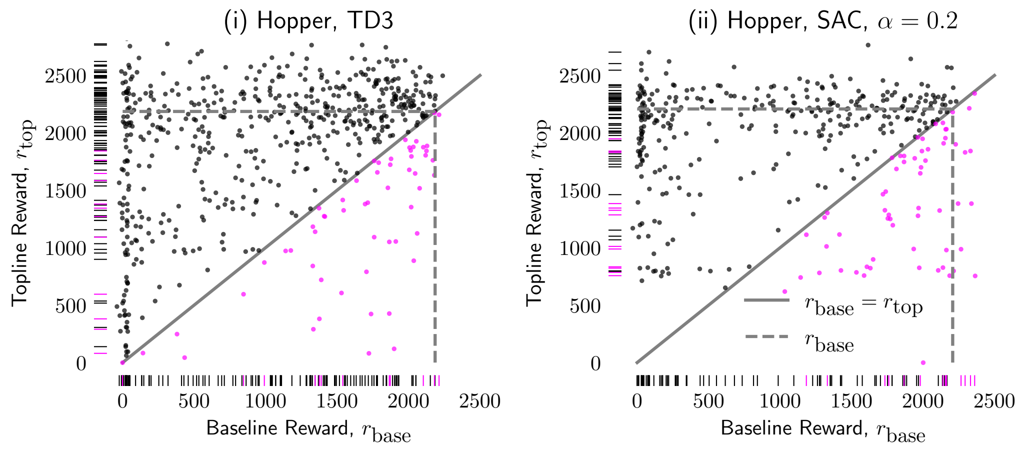


Figure 5: Comparison of baseline and topline reward that can be obtained on the perturbed environments. Ideally, topline models are strictly better (black) than baseline models. 9.5% of all runs points violate this condition for TD3, 15.25% violate the condition for SAC (pink).

B.3 Detailed results, Fig. 2: Environment mismatch correlates with degradation in reward

We show the full results underlying Fig. 2 in Fig. 6, including results for TD3 and SAC across all values of α. Performance generally degrades for large α values with exception of the Hopper agent; besides the performance degradation the qualitative relationship is rather independent of the exact value of α.

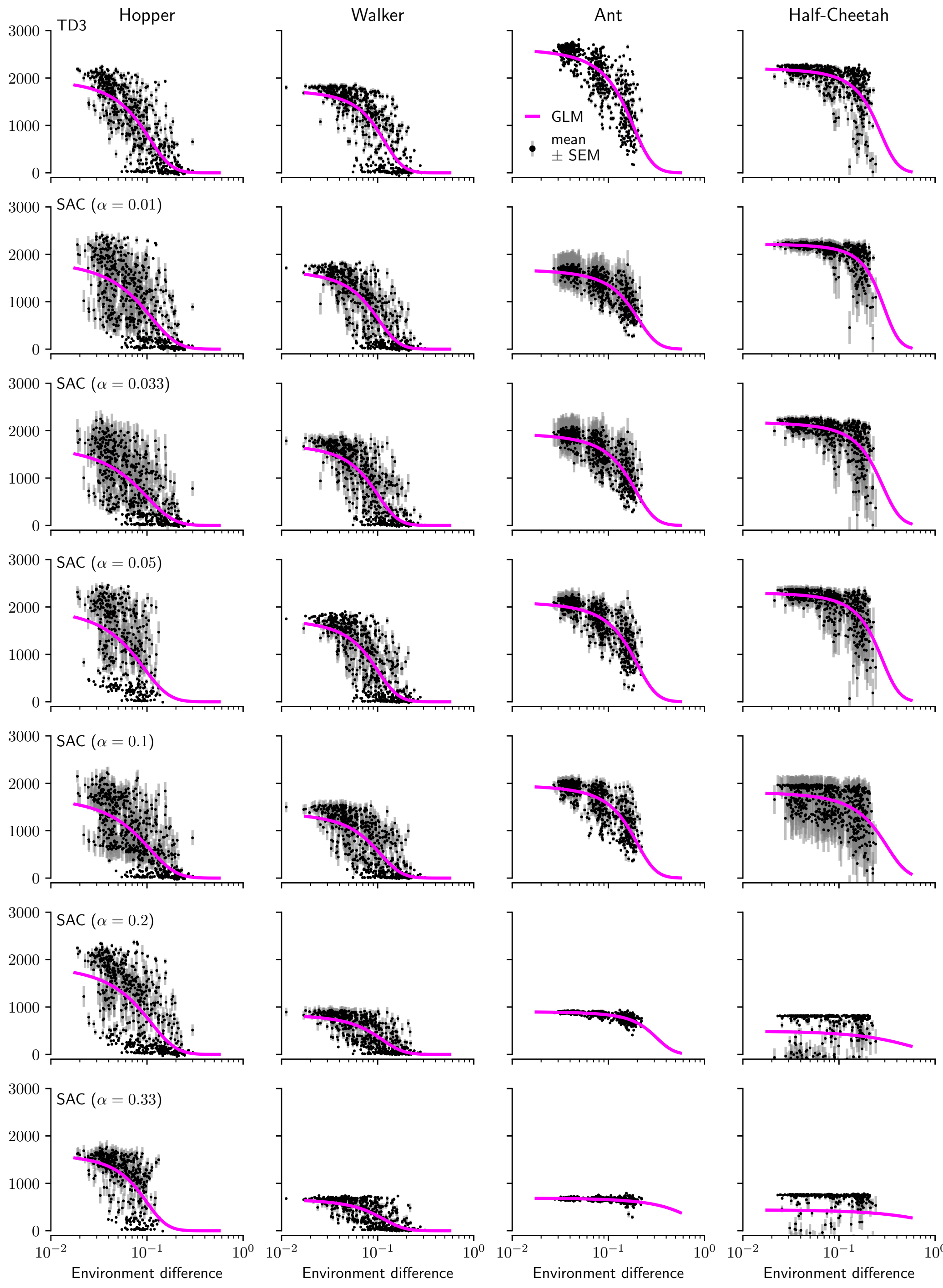


Figure 6: We show detailed results for TD3 and all SAC models, extending Fig. 2.

B.4 DETAILED RESULTS, FIG. 3: BASELINE CORRELATES WITH OUT-OF-DISTRIBUTION REWARD

We show the full results underlying Fig. 3, including results for TD3 and SAC across all values of α, in Fig. 7.

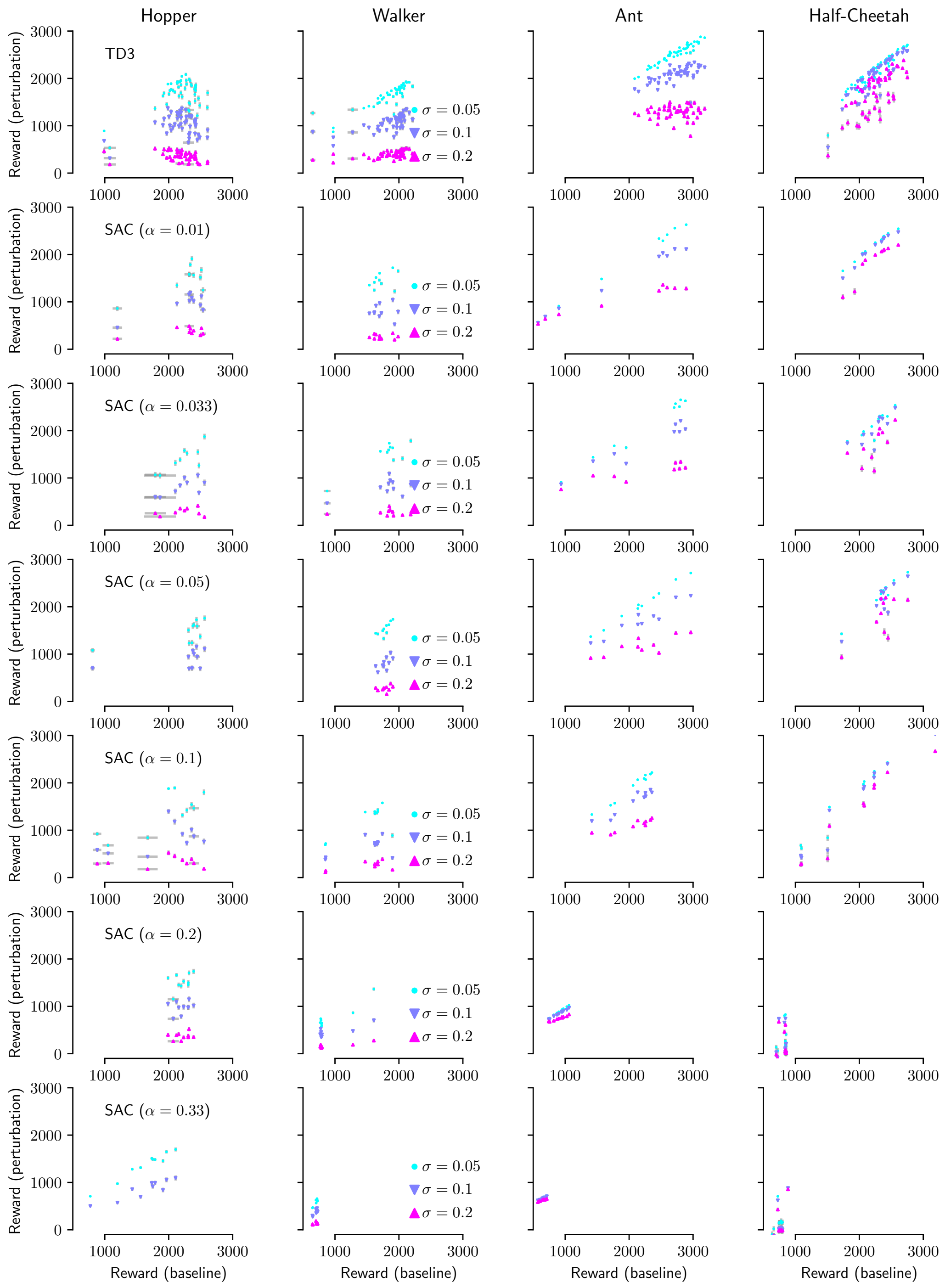


Figure 7: We show detailed results for TD3 and all SAC models, extending Fig. 3.

B.5 Detailed results, Fig. 4: Out-of-distribution error of self-supervised learning predicts drop in reward

We show the full results underlying Fig. 4, including results for TD3 and SAC across all values of α. Results for self-supervised models trained on states only are depicted in Fig. 8, results for self-supervised models trained on state action pairs as in the paper are depicted in Fig. 9. For all models except Hopper (the only environment where $\alpha = 0.2$ attained good performance, cf. Tbl 3), the relationship breaks for sub-optimal values of $\alpha > 0.1$ and is stable in all other regions.

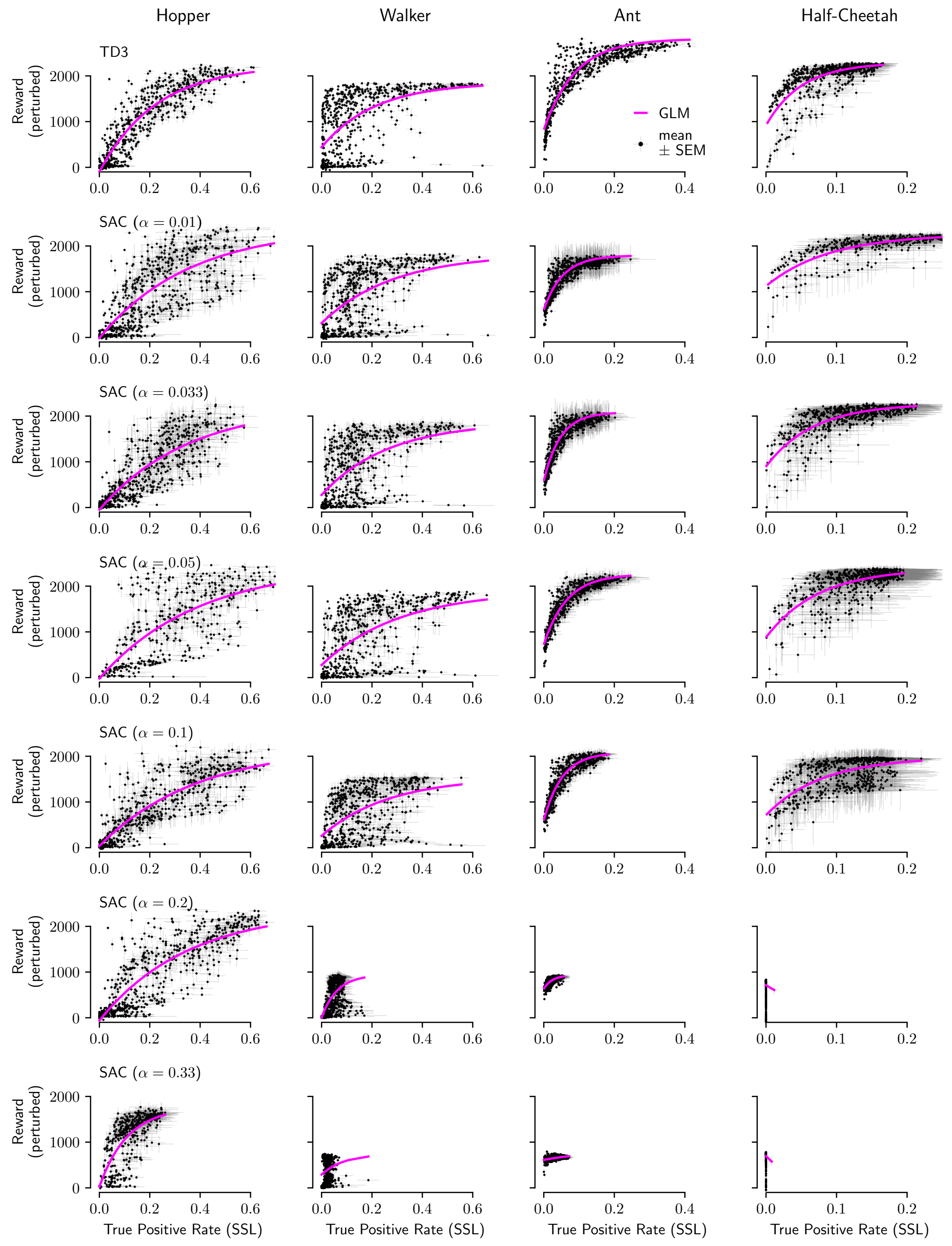


Figure 8: Full results for SSL vs. RL performance. In contrast to Fig. 4, the SSL model is trained on states only.

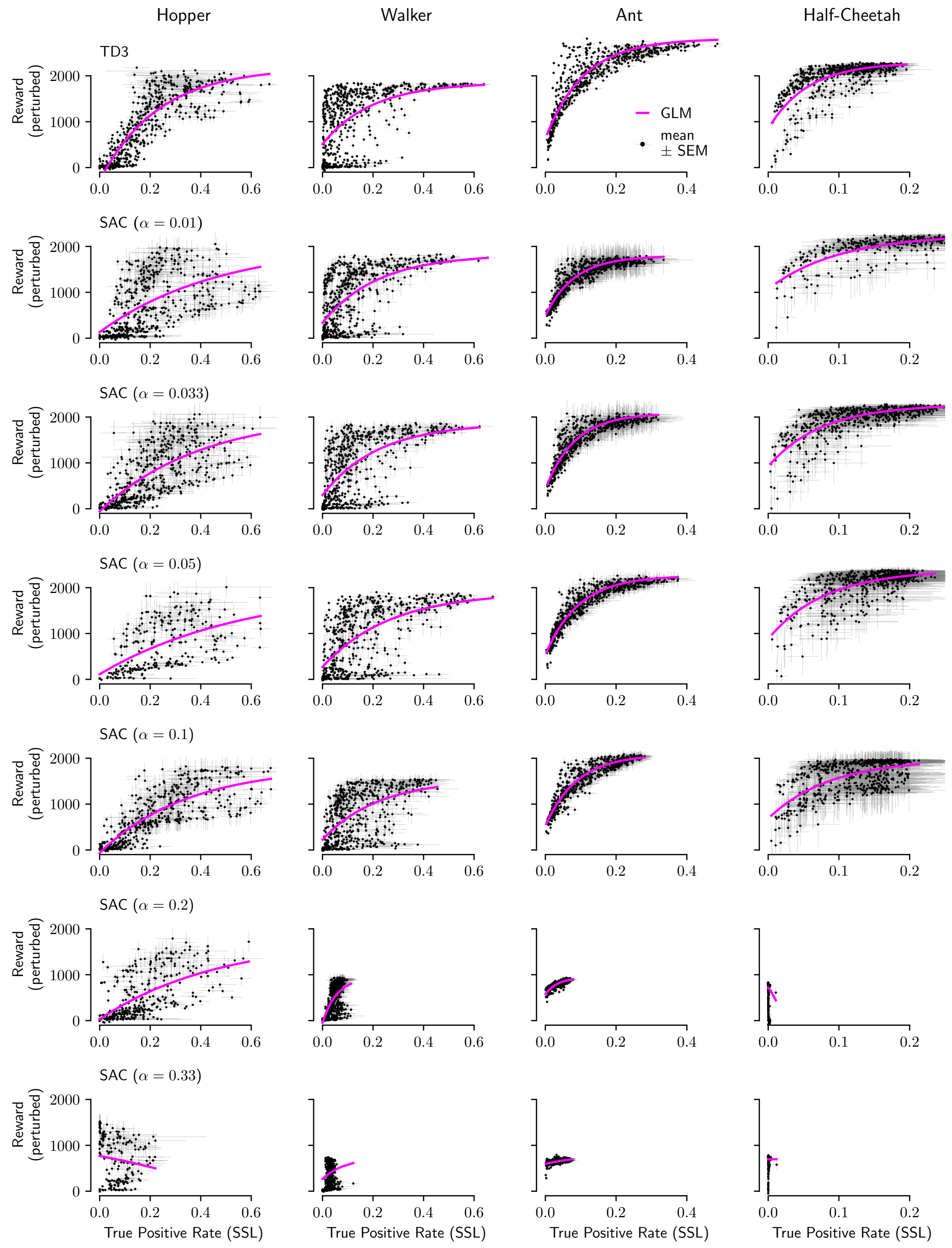
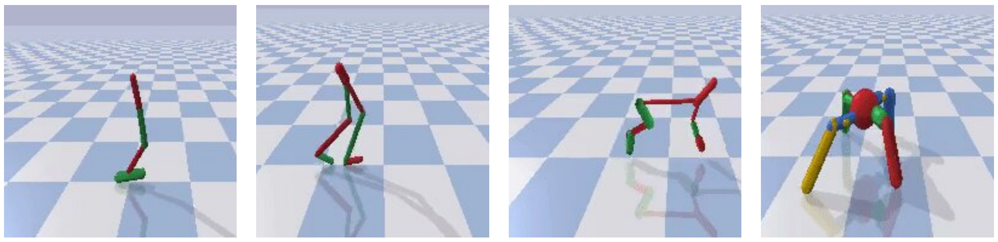


Figure 9: Full results for SSL vs. RL performance. Fig. 4 shows a part of this figure.

C ENVIRONMENTS



(a) Hopper (b) Walker (c) Half-Cheetah (d) Ant

Figure 10: Continuous control agents in PyBullet physics engine

Table 4: Default values of Hopper model

Model parameters	Width	Length
Torso	0.05	1.45
Thigh	0.05	1.05
Leg	0.04	0.6
Foot	0.06	0.26

Table 5: Default values of Walker model

Model parameters	Width	Length
Torso	0.05	1.45
Thigh	0.05	1.05
Leg	0.04	0.6
Foot	0.06	0.2

Table 6: Default values of Ant model

Model parameters	Width	Length
Torso	0.25	0.25
Front left leg joint1	0.08	0.2
Front left leg joint2	0.08	0.2
Front left leg foot	0.08	0.4
Front right leg joint1	0.08	0.2
Front right leg joint2	0.08	0.2
Front right leg foot	0.08	0.4
Left back leg joint1	0.08	0.2
Left back leg joint2	0.08	0.2
Left back leg foot	0.08	0.4
Right back leg joint1	0.08	0.2
Right back leg joint2	0.08	0.2
Right back leg foot	0.08	0.4

Table 7: Default values of Half-Cheetah model

Model parameters	Width	Length
Torso	0.046	0.5
Head	0.046	0.15
Front thigh	0.046	0.145
Front shin	0.046	0.15
Front foot	0.046	0.094
Front thigh	0.046	0.133
Front shin	0.046	0.106
Front foot	0.046	0.07

Hopper ($\boldsymbol{s} \in \mathbb{R}^{15}$, $\boldsymbol{a}A \in \mathbb{R}^{3}$) is an 3 DoF two-dimensional one-legged—and hence, unstable—agent. The environment is reset when the model falls over. The default values of different Hopper limbs in the PyBullet simulator are listed in Tbl. 4.

Walker ($\boldsymbol{s} \in \mathbb{R}^{22}$, $\boldsymbol{a} \in \mathbb{R}^{6}$) is a 6-DoF two-legged agent resembling two connected instances of the Hopper model. The goal to move forward is achieved by walking motion. The environment is reset when the model falls over. Default values of Walker limbs are listed in Tbl. 5.

Half-Cheetah ($\boldsymbol{s} \in \mathbb{R}^{26}$, $\boldsymbol{a} \in \mathbb{R}^{6}$) is a 6-DoF two-dimensional agent. The goal to move forward is achieved by running. The default values for this agent used in simulation are listed in Tbl. 7.

Ant ($\boldsymbol{s} \in \mathbb{R}^{28}$, $\boldsymbol{a} \in \mathbb{R}^{8}$) is an 8-DoF three-dimensional four-legged agent. The agent uses alternate legs at a time for performing forward locomotion. The default limbs and their values for this agent are listed in Table 6.