Objective Robustness in Deep Reinforcement Learning

Jack Koch [*] **Lauro Langosco** [*] **Jacob Pfau** **James Le** **Lee Sharkey**

Abstract

We study *objective robustness* failures, a type of out-of-distribution robustness failure in reinforcement learning (RL). Objective robustness failures occur when an RL agent retains its capabilities out-of-distribution yet pursues the wrong objective. This kind of failure presents different risks than the robustness problems usually considered in the literature, since it involves agents that leverage their capabilities to pursue the wrong objective rather than simply failing to do anything useful. We provide the first explicit empirical demonstrations of objective robustness failures and present a partial characterization of its causes.

1. Introduction

Out of distribution (OOD) robustness, performing well on test data that is not distributed identically to the training set, is a fundamental problem in machine learning (Arjovsky, 2021). This is crucial since in many applications it is not feasible to collect data distributed identically to that which the model will encounter in deployment.

In this work, we focus on a particularly concerning type of OOD robustness called *objective robustness* (Hubinger, 2020a) which we study in the reinforcement learning (RL) setting. Usually, when an RL model is deployed out-of-distribution, the model either performs well or simply fails to take useful actions. However, there exists an alternative failure mode in which the agent pursues an objective other than the training reward while retaining all or most of the capabilities it had on the training distribution. We call this kind of failure an *objective robustness* failure and distinguish it from *capability robustness* failures. To highlight and illustrate this class of failures, we provide empirical demonstrations of the phenomenon in deep RL agents—to our knowledge, this is the first time it has been demonstrated empirically.

[*]Equal contribution . Correspondence to: Jack Koch <jack@jbkjr.com>, Lauro Langosco <langosco.lauro@gmail.com>.

Presented at the ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning., Copyright 2021 by the author(s).

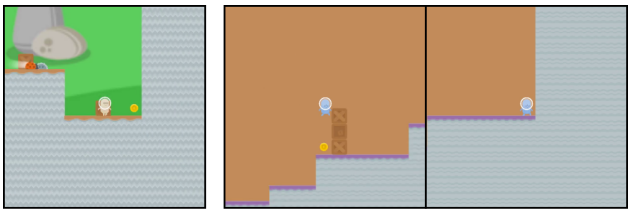

Figure 1. An agent trained to collect the coin reliably reaches reaches the coin during training (left) but frequently skips the coin to reach the end of the level when the coin position is randomized at test time (right).

While capability robustness failures are concerning, objective robustness failures are potentially more dangerous, since an agent that capably pursues an incorrect objective can leverage its capabilities to visit arbitrarily bad states. Our main contributions are:

- We highlight the class of objective robustness failures, differentiate it from other robustness problems, and discuss why it is important to address (Section 2).
- We demonstrate that objective robustness failures occur in practice by training deep RL agents on the Procgen benchmark (Mohanty et al., 2021), a set of diverse procedurally generated environments designed to induce robust generalization, and deploying them on modified environments (Section 3).

2. Objective Robustness

It is widely known that deep learning systems may fail in unexpected ways when deployed out-of-distribution; for example, a classifier trained on images of cows in Alpine pastures will typically fail to recognize an image of a cow that is taken on a sandy beach (Beery et al., 2018; Arjovsky, 2021). We focus on the reinforcement learning setting (Sutton & Barto, 2018), in which a system is trained to take actions in order to maximize a reward. OOD robustness problems frequently arise in RL—here are a two examples:

1. A robot is trained in a simulation where data is plentiful. It is then deployed in the real world, where it

encounters subtle differences in physical parameters (Andrychowicz et al., 2020).

2. A factory wants to use deep RL for a process control application. The agent is trained on episodes of fixed length. When it is deployed in the real world, the episode never ends (there are no episode resets in the real world), and the distribution of observations slowly drifts over time as its context in the world changes.

2.1. Defining objective robustness

A deep RL agent is trained to maximize reward $R\colon S \times A \times S \to \mathbb{R}$, where S and A are the sets of all valid states and actions, respectively. Assume that the agent is deployed under distributional shift; that is, something about the environment (for example the distribution of states) changes at test time. If the agent now achieves low reward in the new environment because it continues to act capably yet appears to optimize a different objective $R' \neq R$, then we say the agent underwent an *objective robustness failure*. We call R' the *behavioral objective* of the agent.

We must specify what counts as a behavioral objective (for example, we wish to avoid ascribing a behavioral objective to an agent that simply acts randomly). To do this, we understand the behavioral objective as equivalent to the notion of a "goal" under the *intentional stance* (Dennett, 1989): a system has one if describing the system as pursuing it is useful for predicting that system's behavior. This rules out e.g. an objective that is simply the indicator function for the policy. In particular, we do not assume that the behavioral objective is represented explicitly by the agent, although this may sometimes be the case.

Our primary motivation for identifying this class of failures is that objective robustness failures—failures in which systems take competent yet misaligned actions when deployed out-of-distribution—have the potential to be particularly dangerous. Other types of failures are bounded in the damage they can do: the worst that can happen is an accident (say, a self-driving car failing to brake). In contrast, an agent that pursues an incorrect objective can perform arbitrarily badly relative to the true reward and indeed becomes more dangerous the more capable it is.

2.2. Properties and causes of objective robustness failures

When should we expect models to to be objective robust? We begin by identifying prerequisites for objective robustness failure:

1. The training environment must be diverse enough to learn sufficiently robust capabilities.
2. There must exist some proxy $R' : S \times A \times S \to \mathbb{R}$ that approximately tracks the true reward on the training distribution.
3. The proxy and the true reward come apart on the OOD test environment.

While these are plausible necessary conditions for objective robustness failure, they are by no means sufficient since, by themselves, they do not guarantee that the model learns to follow the proxy reward R' instead of the true reward. However, assumptions (1) and (2) are also very weak: almost every real-world problem requires a diverse training environment (to achieve capability robustness), and proxies are common in complex environments. Thus objective robustness failure depends mostly on whether the inductive biases of the model prime it to learn a proxy that then (3) diverges from the true objective on the test set.

What proxies then do models tend to learn? Following Hubinger et al. (2019), we (non-exhaustively) list factors that increase the likelihood of learning a particular proxy R'.

- The proxy R' is simpler than the true reward R.
- The true reward R is sparse, but the proxy R' is dense (Pezeshki et al., 2020).
- The true reward R is computationally hard to predict without using the proxy, while the proxy is easily predictable.
- If R is hard to predict, then large model capacity may make objective robustness failure *less* likely, because the model is better able to model the true reward.

As an example for such proxies, consider human evolution. While very different from SGD, biological evolution is (to an approximation) a local optimization process that maximizes inclusive genetic fitness. Humans, however, generally have no desire to maximize the number of our descendants. Instead, we pursue objectives which in the ancestral environment were good proxies for fitness, such as friendship, food, and love. This illustrates a general phenomenon: given a challenging objective, complex environments are rife with proxies for and sub-goals of that objective, many of which are more dense or easier to predict. In addition, human evolution illustrates how such proxies can come apart from the true objective under distributional shift: in the ancestral environment humans were driven to eat high-caloric foods, which in the modern world often leads to obesity. We illustrate this sort of shift in a simple experiment in Section 3.3 that relies on sub-goals. This suggests that objective robustness will be a problem in complex, real-world tasks.

3. Experiments

In this section we provide simple empirical demonstrations of objective robustness failures.[1] For example, a natural interpretation of our results in CoinRun in Section 3.1 is that the agent has learned a *robust capability* to avoid obstacles and navigate the levels,[2] but a *non-robust objective*, since its behavior out-of-distribution is better described by "get to the end of the level" than "go to the coin".

Different kinds of failure The experiments illustrate different flavors of objective robustness failures. *Action space proxies* (CoinRun): the agent substitutes a simple action space proxy ("move right") for the true reward, which could have been identified in terms of a simple feature in its input space (the yellow coin). *Observation ambiguity* (Maze): The observations contain multiple features that identify the goal state, which come apart in the OOD test distribution. *Instrumental goals* (Keys and Chests): The agent learns an objective (collecting keys) that is only instrumentally useful to acquiring the true reward (opening chests).

3.1. CoinRun

In CoinRun, a platformer, the agent spawns on the left side of the level and has to avoid enemies and obstacles to get to a coin (the reward) at the far right of the level. To induce an objective robustness failure, we create a test environment in which coin position is randomized (but accessible).

The agent is trained on vanilla CoinRun and deployed in the modified test environment. At test time the agent generally ignores the coin completely. While the agent sometimes runs into the coin by accident, it often misses it and proceeds to the end of the level, as shown in Figure 1. In terms of Section 2, the agent's behavioral objective is not to collect the coin, but to reach the end of the level.

3.2. Maze

3.2.1. Variant 1

We modify the Procgen Maze environment in order to implement an idea from Hubinger (2020b). A maze is generated using Kruskal's algorithm (Kruskal, 1956), and the agent is trained to navigate towards a piece of cheese located at a random spot in the maze. Instead of training on the original environment, we train on a modified version in which the cheese is always located in the upper right corner (Figure 2).

When deployed in the original Maze environment at test time, the agent does not perform well; it ignores the randomly placed objective, instead navigating to the lower right corner of the maze as usual. The training objective is to reach the cheese, but the behavioral objective of the learned policy is to navigate to the lower right corner.

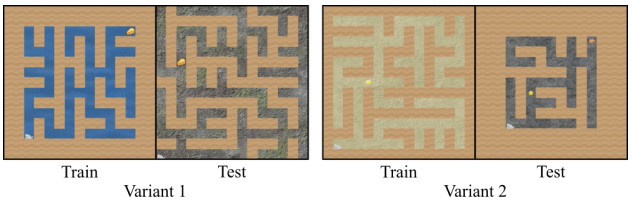

Figure 2. **Left:** The reward (a piece of cheese) is always in the top right of the maze. When cheese is placed randomly, it ignores it and goes to the top right of the maze. **Right:** An agent is trained to navigate to a yellow gem. In the test environment, it must choose between a yellow star and a red gem, and it consistently navigates to the yellow star.

3.2.2. Variant 2

We hypothesize that in CoinRun, the policy that always navigates to the end of the level is preferred because it is simple in terms of its action space: simply move as far right as possible. The same may be true for the Maze experiment in Section 3.2.1, where the agent has learned to navigate to the top right corner. In both experiments, the objective robustness failure arises because a visual feature (coin / cheese) and a positional feature (right / top right) come apart at test time, and the inductive biases of the model favor the latter. However, objective robustness failures can also arise due to other kinds of distributional shift. To illustrate this, we present a simple setting in which there is no *positional* feature that favors one objective over the other; instead, the agent is forced to choose between two ambiguous visual cues.

We train an RL agent on a version of the Procgen Maze environment where the reward is a randomly placed *yellow gem*. At test time, we deploy it on a modified environment featuring two randomly placed objects: a yellow *star* and a *red* gem; the agent is forced to choose between consistency in shape or in color (Figure 2). Except for occasionally getting stuck in a corner, the agent almost always successfully pursues the yellow star, thus generalizing in favor of color rather than shape consistency.

3.3. Keys and Chests

This environment, which we implement by adapting Heist from Procgen, is a maze with two kinds of objects: keys and chests. Whenever the agent comes across a key it is added to a key inventory. If it subsequently comes across a chest,

[1] Video examples of objective robustness failures in all of the following environments can be found here. The code for the modified environments is available from this GitHub repository.

[2] After all, Procgen (Mohanty et al., 2021) was designed to test (capability) generalization in deep RL.

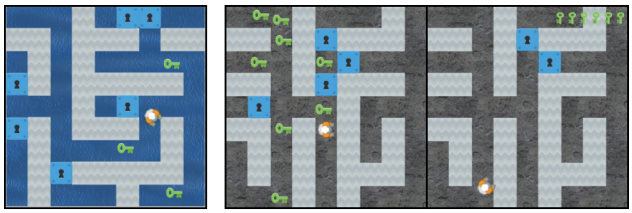

Figure 3. Objective robustness failure on the "keys and chests" task. The agent must collect keys in order to open chests and is only rewarded for opening chests. **Left:** The agent is trained on procedurally generated mazes in which there are twice as many chests as keys. **Right:** At test time, there are instead twice as many keys as chests. The agent more highly values collecting keys than opening chests; it routinely goes out of the way to collect all the keys before opening any remaining chests despite the fact that doing so offers no benefit to its actual return

the chest is opened and a key is deleted from the inventory. The agent is rewarded for every chest it opens.

The objective robustness failure arises due to the following distributional shift between training and test environments: in the training environment, there are twice as many chests as keys, while in the test environment there are twice as many keys as chests. The basic task facing the agent is the same (reward is only given upon opening a chest), but the circumstances are different.

We observe that an agent trained on the "many chests" distribution goes out of its way to collect all the keys before opening the last chest on the "many keys" distribution (Figure 3), even though only half of them are even instrumentally useful for the true reward. Applying the intentional stance, we describe the agent as having learned a simple behavioral objective: collect as many keys as possible, while sometimes visiting chests.

This strategy leads to high reward in an environment where chests are plentiful and the agent can thus focus on looking for keys. However, this proxy objective fails under distributional shift when keys are plentiful and chests are no longer easily available.

4. Related Work

Reward misspecification Reward specification is the problem of specifying a reward that captures the behavior we want (Amodei et al., 2016; Clark & Amodei). Objective robustness is a distinct problem: it may still fail even if the reward function is perfectly specified.[3]

Out-of-Distribution Robustness Objective robustness is a type of OOD robustness. Here, the goal is to optimize worst-case performance over a perturbation set of possible test domains (environments). Causes for this type of train-test mismatch (non-exhaustively) include 1) the training data does not characterize the true distribution (Torralba & Efros, 2011), 2) the distribution shifts over time (Quiñonero-Candela et al., 2009), and 3) the training or the test data are adversarially perturbed (Huang et al., 2017; Gleave et al., 2021; Kos & Song, 2017). Approaches that address OOD robustness include learning causally invariant representations (Arjovsky et al., 2020; Krueger et al., 2021) and modifying the training objective (Patrini et al., 2017).

Unidentifiability Objective robustness failures tend to arise when there are multiple possible reward functions that are indistinguishable from the true reward. This type of unidentifiability is analogous to the one encountered in inverse reinforcement learning (IRL). For example, in their seminal paper on IRL, Ng & Russell (2000) note that identifying the exact reward function an optimal agent optimizes with its behavior is in general impossible.

5. Discussion

We have provided the first (to our knowledge) explicit examples of objective robustness failures in deep RL systems. While deep RL practitioners may be aware that this type of failure is possible, there has not yet been much discussion of objective robustness as distinct from other types of out-of-distribution robustness. We argue that objective robustness is a natural category since, much like adversarial robustness failures, objective robustness failures have distinct causes and pose distinct problems. By introducing the objective robustness problem to a broader audience, we hope to spark interest in it as an avenue for future research.

5.1. Limitations

- Our experiments demonstrate the existence of objective robustness failures and illustrate some of their causes; they do not prove that they will occur in problems of interest. We do however think that there are good arguments for this (Section 2.2).

- In time, the objective vs. capability robustness distinction may be superseded by a categorization that better captures the problems that occur in practice, is more grounded in theory, or both. However, there are different kinds of possible robustness failures, and understanding their varying causes and consequences is important for building safe and capable AI systems.

[3]Failures due to objective misspecification occur when the model behaves in an unintended way that nevertheless scores *highly* on the reward function. In contrast, in failures of objective robustness, models score *poorly* on the training reward because they are pursuing an different objective.

Acknowledgements

Special thanks to Rohin Shah and Evan Hubinger for their guidance and feedback throughout the course of this project. Thanks also to Max Chiswick for assistance adapting the code for training the agents, Adam Gleave for helpful feedback on drafts of this paper, and the organizers of the AI Safety Camp for bringing the authors of this paper together: Remmelt Ellen, Nicholas Goldowsky-Dill, Rebecca Baron, Max Chiswick, and Richard Möhn.

This work was supported by funding from the AI Safety Camp and from the Open Philantropy Project.

References

Amodei, D., Olah, C., Steinhardt, J., Christiano, P. F., Schulman, J., and Mané, D. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Arjovsky, M. *Out of Distribution Generalization in Machine Learning*. PhD thesis, New York University, 2021.

Arjovsky, M., Bottou, L., Gulrajani, I., and Lopez-Paz, D. Invariant risk minimization, 2020.

Beery, S., Van Horn, G., and Perona, P. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 456–473, 2018.

Clark, J. and Amodei, D. Faulty reward functions in the wild. OpenAI Blog. URL https://openai.com/blog/faulty-reward-functions/.

Dennett, D. *The Intentional Stance*. MIT Press, Cambridge, 1989.

Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., and Russell, S. Adversarial policies: Attacking deep reinforcement learning, 2021.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies, 2017.

Hubinger, E. Clarifying inner alignment terminology. AI Alignment Forum, 2020a. URL https://www.alignmentforum.org/posts/SzecSPYxqRa5GCaSF.

Hubinger, E. Towards an empirical investigation of inner alignment. AI Alignment Forum, 2020b. URL https://www.alignmentforum.org/posts/2GycxikGnepJbxfHT.

Hubinger, E., van Merwijk, C., Mikulik, V., Skalse, J., and Garrabrant, S. Risks from learned optimization in advanced machine learning systems. *arXiv preprint arXiv:1906.01820*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kos, J. and Song, D. Delving into adversarial attacks on deep policies, 2017.

Krueger, D., Caballero, E., Jacobsen, J.-H., Zhang, A., Binas, J., Zhang, D., Priol, R. L., and Courville, A. Out-of-distribution generalization via risk extrapolation (rex), 2021.

Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. ISSN 0002-9939, 1088-6826. doi: 10.1090/S0002-9939-1956-0078686-7. URL https://www.ams.org/proc/1956-007-01/S0002-9939-1956-0078686-7/.

Lee, H. Training procgen environment with pytorch. https://github.com/joonleesky/train-procgen-pytorch, 2020.

Mohanty, S., Poonganam, J., Gaidon, A., Kolobov, A., Wulfe, B., Chakraborty, D., Šemetulskis, G., Schapke, J., Kubilius, J., and Pašukonis, J. Measuring Sample Efficiency and Generalization in Reinforcement Learning Benchmarks: NeurIPS 2020 Procgen Benchmark. *arXiv preprint arXiv:2103.15332*, 2021.

Ng, A. and Russell, S. Algorithms for inverse reinforcement learning. *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., and Qu, L. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1944–1952, 2017.

Pezeshki, M., Kaba, S.-O., Bengio, Y., Courville, A., Precup, D., and Lajoie, G. Gradient starvation: A learning proclivity in neural networks. *arXiv preprint arXiv:2011.09468*, 2020.

Quiñonero-Candela, J., Sugiyama, M., Lawrence, N. D., and Schwaighofer, A. *Dataset shift in machine learning*. Mit Press, 2009.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Torralba, A. and Efros, A. A. Unbiased look at dataset bias. In *CVPR 2011*, pp. 1521–1528. IEEE, 2011.

A. Varying how often the coin is randomly placed in training

To test how stable the objective robustness failure is, we train a series of agents on environments which vary in how often the coin is placed randomly. We then deploy those agents in the test environment in which coin position is always randomized. Results can be seen in Figure 4, which shows the frequencies of two different outcomes, 1) failure of capability: the agent dies or gets stuck, thus neither collecting the coin nor to the end of the level, and 2) failure of objective: the agent misses the coin and navigates to the end of the level. As the diversity of the training environment increases, the proportion of objective robustness failures decreases.

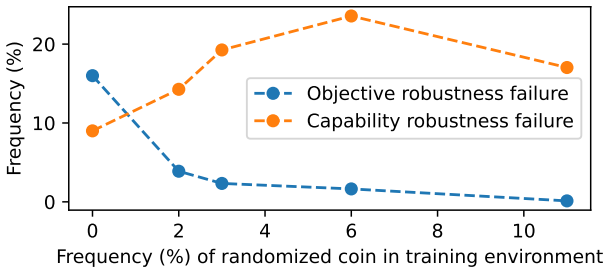

Figure 4. How diverse does the training distribution need to be to induce objective robustness? We train agents in a CoinRun environment in which the coin is placed randomly $\{0, 1, 2, 5, 10\}\%$ of the time, and keep track of how often the agent navigates to the end of the level while ignoring the coin.

B. Implementation

All environments are adapted from the Procgen benchmark (Mohanty et al., 2021). This benchmark is built to study sample efficiency and generalization to within-distribution tasks. Agents are tasked with performing well in an arcade-like video game from pixel observations. The environments are procedurally generated and thus diverse; to perform well, an agent is forced to learn strategies that work in a wide range of task settings and difficulty and cannot rely on e.g. memorizing a small number of trajectories to solve a fixed set of levels.

For all environments, we use an Actor-Critic architecture using Proximal Policy Optimization (PPO) (Schulman et al., 2017). Hyperparameters can be found in Table 1. Code to reproduce the experiments may be found here. All models are implemented in PyTorch (Paszke et al., 2019), and our implementation is based on a codebase by Lee (2020). Unless otherwise stated, models are trained on 100k procedurally generated levels for 200M timesteps. Each training run required approximately 30 GPU hours of compute on a V100.

B.1. Hyperparameters

We use the Adam optimizer (Kingma & Ba, 2015) in all experiments.

Table 1. PPO Hyperparameters

ENV. DISTRIBUTION MODE	HARD
γ	.999
λ	.95
LEARNING RATE	0.0005
# TIMESTEPS PER ROLLOUT	256
EPOCHS PER ROLLOUT	3
# MINIBATCHES PER EPOCH	8
MINIBATCH SIZE	2048
ENTROPY BONUS (k_H)	.01
PPO CLIP RANGE	.2
REWARD NORMALIZATION?	YES
LEARNING RATE	5×10^{-4}
# WORKERS	4
# ENVIRONMENTS PER WORKER	64
TOTAL TIMESTEPS	200M
ARCHITECTURE	Impala
LSTM?	No
FRAME STACK?	No