# Reward-guided Curriculum for Learning Robust Action Policies

Mysore, Siddharth [1] Platt, Robert [2] Saenko, Kate [1]

## Abstract

We propose a novel method to develop robust action policies using an automated curriculum which seeks to improve task generalization and reduce policy brittleness by self-reflectively choosing what to train on in order to maximize rewards over a task domain. Our Reward-guided Curriculum (RgC) is a single-policy meta-learning approach which is designed to augment the training of existing architectures. Experiments on multiple video games and classical controls tasks indicate notable improvements in task generalization and robustness of the policies trained with RgC.

## 1. Introduction

Deep Reinforcement Learning (RL) techniques have demonstrated laudable performance across a wide array of tasks in games, controls, and more. However, the brittleness and capacity of generalization of trained RL policies remain open problems (Nichol et al., 2018; Leike et al., 2017; Cobbe et al., 2018; Packer et al., 2018; Zhang et al., 2018a;b). These problems are especially important to address in practical applications where one might expect varying degrees of distribution shift at test time, where RL agents may encounter states never experienced during training. Previous works addressing the sensitivity and generalizability of RL policies (Finn et al., 2017; Peng et al., 2018; Gupta et al., 2018; Stadie et al., 2018; Rusu et al., 2016; Yu et al., 2017; Tobin et al., 2017; Sadeghi et al., 2017; Rajeswaran et al., 2017) typically sample from a uniform or normal distribution over tasks. In this work we ask: Is this naive sampling over task variation the best we can do?

In an effort to more intelligently utilize experience gained on a training set and improve robustness and generalization, we develop an approach called *Reward-guided Stochastic Curriculum* (RgC). RgC automatically constructs a training curriculum for each learner to best motivate the learning of robust action policies. We accomplish this by formulating the training curriculum as a multi-armed bandit problem, which seeks to maximize episodic rewards over the training set, with the bandits guiding sampling probabilities over task distributions. Experiments over multiple tasks show that our method helps agents learn more robust policies, which generalize better and are less brittle to distributional shifts in the task domain.

[1]Department of Computer Science, Boston University, Boston, MA, U.S.A [2]Khoury College of Computer Science, Northeastern University, Boston, MA, U.S.A. Correspondence to: Siddharth Mysore <sidmys@bu.edu>.

*Proceedings of the Workshop on Multi-Task and Lifelong Reinforcement Learning at the 36th International Conference on Machine Learning*, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

## 2. Reward-guided Stochastic Curriculum

### 2.1. Preliminaries

A key hypothesis driving the development of the framework for RgC was that not all experiences carry equal significance in training. When a person attempts to learn a new skill, they may not remember every aspect of the learning process, but rather focus on the moments that offered critical insight that allows them to advance their understanding of the skill. We attempt to extend a similar intuition to learning in RL, so that the learning algorithms can automatically identify and prioritize experiences which offer the most value gain.

A number of meta-learning techniques recognize the inequality in the importance of different experiences during training and attempt to more intelligently utilize past experience and knowledge of past tasks when updating policies during training (Andrychowicz et al., 2017; Rajeswaran et al., 2017; Schaul et al., 2016; Finn et al., 2017). However, these methods typically limit their considerations to either past experiences or future task choices. Curriculum learning has also been demonstrated as a powerful strategy in improving learning performance over multiple tasks (Bengio et al., 2009; Oh et al., 2017; Andreas et al., 2016), however curricula are often partially or completely hand crafted.

We propose RgC as a way to holistically consider both the past and future impact of choosing to train on any particular instantiation of a task, relative to the available training set, in order to actively inform how tasks are sampled from the training distribution.

## 2.2. Method

RgC draws inspiration from Graves et al. (2017), who tackle the problem of multi-task Natural Language Processing with automated curricula. Like them, we formulate the problem of developing an automated curriculum for a given RL task as a multi-armed bandit problem. The goal of the bandit is to maximize the payoff of every choice it makes, where the bandit's choices correspond to determining how tasks should be sampled from the task distribution. This would be trivial if the values of each arm are known; however, when choice-values are not known, it is necessary to estimate the value by exploring the task domain.

We define a curriculum as a sampling policy over the task distribution. A basic curriculum over $N$ possible task settings can be constructed as an $N$-armed bandit, with the syllabus of the developed curriculum intended to maximize the reward that the RL agent achieves over the entire task distribution. Over $T$ rounds of 'play', the bandit/curriculum makes a choice, $a_t \in \{1, \ldots, N\}$, corresponding to a decision to train under a specific task setting, and observes a payoff $r_t$, computed as the difference in mean rewards observed before and after training on the selected task setting. The goal of the curriculum is to consistently sample rollouts from the task distribution to maximize learning gains.

To choose settings on which to train in order to minimize regret and maximize overall reward, we employ the Exponentially-weighted algorithm for Exploration and Exploitation (Exp3) (Auer et al., 2003). Specifically, we use the Exp3.S variant of the algorithm to develop our multi-armed bandit's policy, which employs an $\epsilon$-greedy strategy and additively mixes weights to ensure that the probabilities of selecting any particular action are not driven to insignificance. We define $\epsilon$ to limit the maximum probability of any setting being selected. (*Note*: we present Exp3.S similarly to Graves et al. (2017), which is mathematically equivalent to the algorithm as it is presented in Auer et al. (2003)).

A key difference in our method from that employed by Graves et al. (2017) is in how we define the payoff, or the value gained by training on a specific task setting. Graves et al. (2017) perform a comparison on the training loss before and after training, utilizing the same loss metric that is employed by the network. We instead compute our payoff based on the difference in mean episodic rewards over the task distribution before and after training. We further extend the method to attach a different bandit to each different task variable in the training distribution (for example, if mass and volume were variables, we would use two bandits - one for mass and one for volume, and each bandit would select different masses or volumes to train on respectively). This multi-multi-armed bandit structure allows for a linear growth of the number of arms to be maintained with the number of variants per variable, as opposed to polynomial.

A bandit $m \in \boldsymbol{M}$ (where $\boldsymbol{M}$ is a set of task variables) has a policy defined by weights, $w_{m,i}\ \forall i \in \{1, \ldots, N_m\}$, corresponding to the $N_m$ possible task-variable settings. At bandit-step $t$ the bandit chooses setting $a_{m,t} \sim \pi^{\text{Exp3.S}}_{m,t}$, where $\pi^{\text{Exp3.S}}_{m,t}(i_m)$ is the sampling probability of setting $i_m$:

$$\pi^{\text{Exp3.S}}_{m,t}(i) := (1-\epsilon)\frac{e^{w_{m,i,t}}}{\sum_j e^{w_{m,j,t}}} + \frac{\epsilon}{N_m} \tag{1}$$

At the end of each bandit step, the weights are updated based on observed payoff, $r_t$:

$$w_{m,t+1,i} := \log\left[(1-\alpha_t)\eta_i + \frac{\alpha_t}{N_m - 1}\sum_{j \neq i} \eta_j\right] \tag{2}$$
$$\text{where } \eta_k = e^{\left(w_{m,t,k} + \hat{r}^{\beta}_{M,t-1,kR}\right)}$$

where $w_{m,1} = 0$, $\alpha_t := t^{-1}$, and the importance sampled payoff is computed as:

$$\hat{r}^{\beta}_{M,t,i} := \frac{r_t \prod_{m \in M} \mathbb{I}_{[a_{m,t} = i_m]} + \beta}{\prod_{m \in M} \pi^{\text{Exp3.S}}_{m,t}(i_m)} \tag{3}$$

To bound the magnitude by which an arm's weight might change at any given step, payoffs, $r_t$, per bandit step, $t$ are scaled such that $r_t \in [-1, 1]$:

$$r_t := \begin{cases} -1 & \delta R_t < \mu^{20}_t \\ 1 & \delta R_t > \mu^{80}_t \\ \frac{2(\delta R_t - \mu^{20}_t)}{\mu^{80}_t - \mu^{20}_t} - 1 & \text{otherwise} \end{cases} \tag{4}$$

where $\delta R_t = R_t - R_{t-1}$ is the true bandit policy payoff at step $t$, computed based on mean rewards achieved by the agent on the set of environment setting of interest, and $\mu^x$ represents the $x^{\text{th}}$ percentile of payoffs achieved: $\{r_{s \leq t}\}$. The complete RgC algorithm can be found in Appendix A.

As its name suggests, the Reward-guided Curriculum is not a RL algorithm in itself, but is rather a meta-learning scheme that is built to automatically guide how RL agents explore the training domain in an effort to improve generalization over the task distribution. As discussed in Section 2.1, it is built on the hypothesis that following a training curriculum which allows the agent to acquire more 'useful' experience would result in more robust performance. The practical realization of this is three-fold:

i Agents always consider the impact of experience gained in the context of how they affect the average achievable rewards over the entire task distribution, which we believe encourages a stronger focus on being generally competent, as opposed to capitalizing on instances of high rewards, which may be gained by specializing on individual task settings.

ii By contextualizing past experiences and periodically updating the sampling strategy over the training distribution, RgC constantly pushes the RL agent to explore

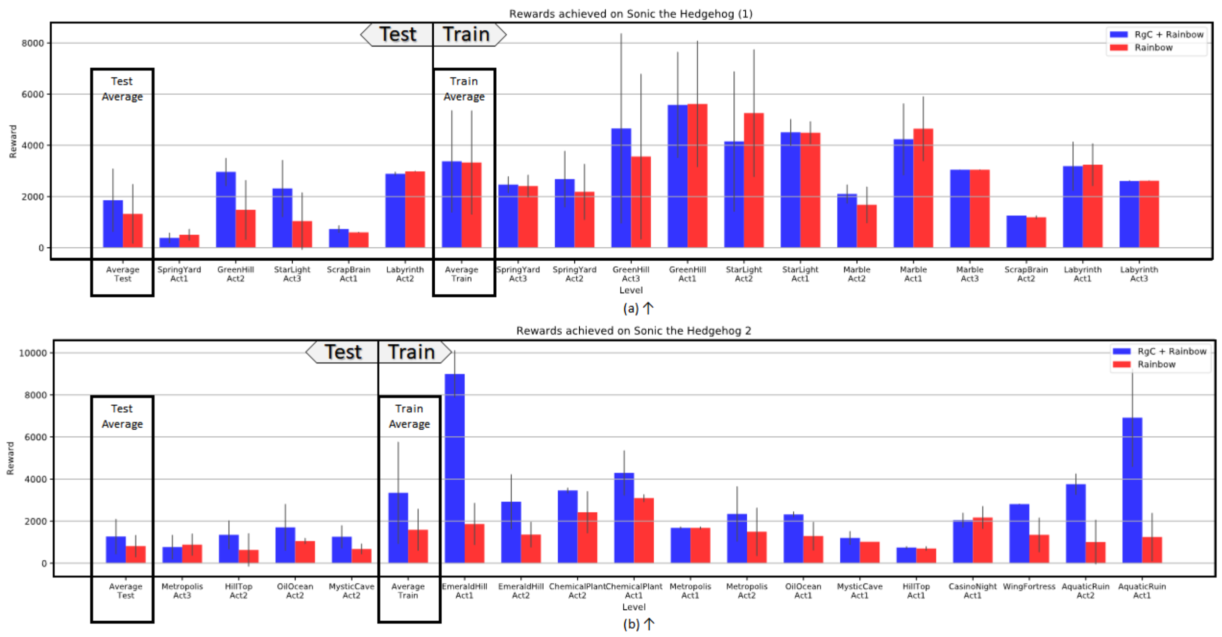

*Figure 1.* Comparison between RgC (blue) and baseline Rainbow (red) of rewards achieved on Sonic the Hedgehog 1 (a) and Sonic the Hedgehog 2 (b). While the test performances for both games are lower than on training levels, RgC performs consistently better than the baseline joint-Rainbow, suggesting that RgC aids the development of better in-domain generalization.

the task-domain in a way that maximizes expected value gain.

iii By always prioritizing the gaining of experiences which contribute to improved generalized performance, RgC also contributes to replay buffers being filled with more potentially valuable state transitions.

## 3. Evaluation

In order to investigate the broader utility of RgC, we investigate the performance impacts of our algorithm on two classes of tasks: (i) playing video games, and (ii) classical controls. Video games offer multiple aesthetically and mechanically similar levels, and the level-based structure of games allows us to create distinct training and test sets on which we can test the generalizability of RL agent policies. Controls tasks on the other hand, which model physical interactions, require agents to be robust to variations and inaccuracies in modeling, providing a good test bed for studying policy brittleness.

### 3.1. Sonic Genesis Games

The Sonic the Hedgehog games were featured as a part of the OpenAI Gym-retro generalizability challenge (Nichol et al., 2018). However, because the contest did not make their 3-game training environment publicly available, we consider generalizability across only 2 of the Sonic games individually, with sets of training and held-out test levels for each game. We otherwise use the same game interfaces, reward engineering and hyperparameters. We train our agents using the Rainbow (Hessel et al., 2017) architecture, which is one of the baselines provided by Nichol et al. (2018). We specifically use the joint-Rainbow baseline, which trains agents by randomly re-sampling the task setting on every new episode (this is similar to domain randomization (Tobin et al., 2017; Nichol et al., 2018)), and compare it against vanilla Rainbow guided by RgC. Here, RgC selects when to train on which levels to maximize rewards over all the training levels (further implementation details in Appendix B.1).

Analyzing the training progress of either game does not provide any immediate indication of a meaningful improvement in performance with RgC, with observed training rewards being of similar magnitude to joint rainbow, but presenting with higher variance (see Appendix B.2). However, when considering the performance on held-out test levels, as shown in Figure 1, it can be noted that the agents trained with RgC perform better on average than those trained with joint Rainbow. It can also be noted that agents trained with RgC tend to perform better over the levels in the training set with worst-case performance not being significantly worse than that of joint Rainbow. In the case of Sonic the Hedgehog 2, we note from Figure 1(b) that the RgC agents appear to significantly outperform the baseline joint-Rainbow, yet this is not reflected in the training reward. We suspect that this is because, during training, the RgC agents focus on levels which promote a general improvement in performance over the training set of levels. The Emerald Hill Zone Act1 and Aquatic Ruins Zone Act1 levels, where the agents already achieved high rewards and did not choose to focus their attention, were rarely sampled and thus their high rewards had little impact on the training reward signals observed by the underlying Rainbow algorithm.

### 3.2. Simple Continuous Controls

We also evaluated RgC on a series of classical continuous controls tasks - (i) pendulum balancing, (ii) balancing a cart-pole system, and (iii) 2D-manipulation of a ball on a plane. Tasks (i) and (ii) are derived from OpenAI Gym (Brockman et al., 2016), while (iii) is a custom environment built using the Unity game engine. In all three of these tasks, we sought to investigate the efficacy of agents trained with RgC on tasks involving changing physics and continuous control, where we had previously observed evidence of policy brittleness (details in Appendix D). The pendulum and ball-pushing tasks see the mass of the pendulum and ball varied respectively, and the cart-pole has variable cart and pole masses.

Our results are compared against: (i) The best results observed via a grid search (oracle) on policies trained exclusively on specific individual task settings (see Appendix D), and (ii) Policies trained under a joint/mixed training structure (joint) as with the sonic games in Section 3.1. All agents are trained using the Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) architecture. When training with RgC, our curriculum chooses physics settings under which to train the agent. Further implementation details on architecture and task set-up are provided in Appendix C.

Tables 1, 2 and 3 demonstrate that our method outperforms policies built on joint sampling, and achieves a performance more comparable to our oracle, with the Pendulum and Cart-Pole getting within 1% and 4% of their oracles' success rates respectively (noting that the Pendulum's oracle achieved a 100% success rate). In the case of the ball-pushing task, where we did not have a binary definition of success, it can be noted that the average error is improved over joint sampling, being within $2\times$ of the oracle's error, as opposed to $3\times$. Our method also has a significantly lower computational cost than the oracle, needing to train only a single policy as opposed to $\prod_{m\in M} N_m$ policies.

Table 1. *Pendulum policy success rate (higher is better, darker is worse)* averaged over 15 agents, with 6 tests per mass per agent.

| Test Mass | Policy | | |
|---|---|---|---|
| | Oracle | Joint (baseline) | **RgC (ours)** |
| 1 | 1.00 | 1.00 | 1.00 |
| 2 | 1.00 | 1.00 | 1.00 |
| 3 | 1.00 | 1.00 | 1.00 |
| 4 | 1.00 | 1.00 | 1.00 |
| 5 | 1.00 | 1.00 | 1.00 |
| 6 | 1.00 | 0.85 | 1.00 |
| 7 | 1.00 | 0.77 | 1.00 |
| 8 | 1.00 | 0.69 | 1.00 |
| 9 | 1.00 | 0.69 | 1.00 |
| 10 | 1.00 | 0.46 | 0.93 |
| Avg | 1.00 | 0.85 | 0.99 |

Table 2. *Cart-pole policy success rate (higher is better, darker is worse)* averaged over 14 agents, with 15 random initializations per test case.

| Cart Mass | Pole Mass | Policy | | |
|---|---|---|---|---|
| | | Oracle | Joint | **RgC (ours)** |
| 1.0 | 0.10 | 1.00 | 0.83 | 1.00 |
| | 0.25 | 0.83 | 0.83 | 1.00 |
| | 0.50 | 1.00 | 0.83 | 1.00 |
| | 1.00 | 0.83 | 0.83 | 0.93 |
| 3.0 | 0.10 | 0.83 | 1.00 | 0.86 |
| | 0.25 | 0.83 | 1.00 | 0.79 |
| | 0.50 | 0.83 | 0.83 | 0.79 |
| | 1.00 | 0.83 | 0.67 | 0.86 |
| 5.0 | 0.10 | 0.83 | 0.33 | 0.57 |
| | 0.25 | 0.83 | 0.33 | 0.64 |
| | 0.50 | 0.83 | 0.17 | 0.64 |
| | 1.00 | 0.67 | 0.17 | 0.64 |
| Avg | | 0.85 | 0.65 | 0.81 |

Table 3. *Ball pushing policy error rate comparisons (lower is better, darker is worse)* averaged over 15 agents and 50 initializtions per test

| Test Mass | Policy | | |
|---|---|---|---|
| | Oracle | Joint | **RgC (ours)** |
| 2 | 0.21 | 0.55 | 0.44 |
| 4 | 0.21 | 0.64 | 0.44 |
| 6 | 0.22 | 0.68 | 0.45 |
| 8 | 0.22 | 0.68 | 0.45 |
| 10 | 0.22 | 0.69 | 0.45 |
| Avg | 0.22 | 0.65 | 0.45 |

## 4. Conclusion

We proposed RgC, a meta-learning technique learning using automatic stochastic curricula, guided by reward signals on a task distribution, to get the most out of an agent's training environment and develop action policies robust to task perturbation. The curricula developed adapt to the experiences of each learner, allowing for a notion of self-reflection and self-correction. We demonstrate that RgC is capable of improving generalization to unseen test levels in two different games, and allows for reduced policy brittleness when faced with physical variations in a series of classical controls tasks.

We do however recognize a few key limitations of our method. Principally, future work would seek to improve the statistical significance of the results presented in Section 3.1 by analyzing performances of more than just 5 agents. Furthermore, the current assumption that environment can be discretized in the manner presented may not hold in general and we are currently investigating techniques which could extend the principles of RgC to continuous variation in task parametrization.

## Acknowledgements

This work was funded by NSF S&AS and DARPA.

## References

Andreas, J., Klein, D., and Levine, S. Modular multi-task reinforcement learning with policy sketches. *CoRR*, abs/1611.01796, 2016.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5048–5058. Curran Associates, Inc., 2017.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003. ISSN 0097-5397. doi: 10.1137/S0097539701398375.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In Bottou, L. and Littman, M. (eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, 2009.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *CoRR*, abs/1606.01540, 2016.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. *CoRR*, abs/1812.02341, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. *CoRR*, abs/1704.03003, 2017.

Gupta, A., Eysenbach, B., Finn, C., and Levine, S. Unsupervised meta-learning for reinforcement learning. *CoRR*, abs/1806.04640, 2018.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, 2017.

Leike, J., Martic, M., Krakovna, V., Ortega, P. A., Everitt, T., Lefrancq, A., Orseau, L., and Legg, S. Ai safety gridworlds. *CoRR*, abs/1711.09883, 2017.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., and Schulman, J. Gotta learn fast: A new benchmark for generalization in RL. *CoRR*, abs/1804.03720, 2018.

Oh, J., Singh, S. P., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. In *ICML*, 2017.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. X. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311.

Rajeswaran, A., Ghotra, S., Ravindran, B., and Levine, S. Epopt: Learning robust neural network policies using model ensembles. *International Conference on Learning Representations*, 2017.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

Sadeghi, F., Toshev, A., Jang, E., and Levine, S. Sim2real view invariant visual servoing by recurrent control. *CoRR*, abs/1712.07642, 2017.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *International Conference on Learning Representations*, 2016.

Stadie, B. C., Yang, G., Houthooft, R., Chen, P., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. The importance of sampling in meta-reinforcement learning. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 9300–9310, 2018.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world.

*IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 23–30, 09 2017. doi: 10.1109/IROS.2017.8202133.

Yu, W., Liu, C. K., and Turk, G. Preparing for the unknown: Learning a universal policy with online system identification. *Robotics: Science and Systems*, 2017.

Zhang, A. X., Ballas, N., and Pineau, J. A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937, 2018a.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, 2018b.

## A. Reward-guided Curriculum Algorithm

**Algorithm 1** Reward-guided Curriculum

```
Initialize: $w_{m,i} = 0 \ \forall\, i \in N_m \ \ \forall\, m \in \boldsymbol{M}$
for $t = 1 \dots T$ do
    Sample $|\boldsymbol{M}|$ task-variable values $a_{m,t} \sim \pi^{\text{Exp3.S}}_{m,t} \ \forall m \in \boldsymbol{M}$
    Sample $K$ task initializations uniformly from a valid space of initializations
    for $k \in K$ do
        Compute Initial Reward of actor-network policy $p_\theta$ on initialization $k$: $R^{pre}_k$
    end for
    Train network $p_\theta$ on $k \in K$
    for $k \in K$ do
        Compute Post-training Reward of network $p_\theta$ on initialization $k$: $R^{post}_k$
    end for
    Compute learning progress $\delta R_t := mean(\{R^{post}_k - R^{pre}_k\} \ \forall\ k \in K)$
    Map $\delta R_t$ to $[-1, 1]$ by (4)
    Update weights $w_{m,i}$ by (2)
end for
```

## B. Sonic Genesis Games – Implementation

### B.1. Technical details

The hyperparameters for the Rainbow agent for the Sonic the Hedgehog games, both Sonic 1 and 2, was based on the retro-baselines released by OpenAI and found at https://github.com/openai/retro-baselines, supplemented by the Rainbow implementation provided by anyrl-py found at https://github.com/unixpickle/anyrl-py.

Modifications had to me be made to the sonic-util and dqn codes from retro-baselines and anyrl-py/algos respectively to accommodate RgC and additional logging. The latter was a superficial change which did not impact training. Current development codes can be made available upon request.

We use the following split of training ant test levels for Sonic the hedgehog:

Training levels: {'SpringYardZone.Act3', 'SpringYardZone.Act2', 'GreenHillZone.Act3', 'GreenHillZone.Act1', 'StarLightZone.Act2', 'StarLightZone.Act1', 'MarbleZone.Act2', 'MarbleZone.Act1', 'MarbleZone.Act3', 'ScrapBrainZone.Act2', 'LabyrinthZone.Act1', 'LabyrinthZone.Act3'}

Test levels: {'SpringYardZone.Act1', 'GreenHillZone.Act2', 'StarLightZone.Act3', 'ScrapBrainZone.Act1', 'LabyrinthZone.Act2'}

We use the following split of training ant test levels for Sonic the hedgehog 2:

Training levels: {'EmeraldHillZone.Act1', 'EmeraldHillZone.Act2', 'ChemicalPlantZone.Act2', 'ChemicalPlantZone.Act1', 'MetropolisZone.Act1', 'MetropolisZone.Act2', 'OilOceanZone.Act1', 'MysticCaveZone.Act1', 'HillTopZone.Act1', 'CasinoNightZone.Act1', 'WingFortressZone', 'AquaticRuinZone.Act2', 'AquaticRuinZone.Act1'}

Test levels: {'MetropolisZone.Act3', 'HillTopZone.Act2', 'OilOceanZone.Act2', 'MysticCaveZone.Act2'}

Each training run uses a different random seed and training was conducted on a PC running Windows 10, with an Intel i7 6850k and Nvidia GTX 1080ti.

For consistency and fairness to all agents, each agent was allowed to train for exactly $2 \times 10^6$ steps. The RgC curriculum was designed to perform 1 curriculum step per 10 episodes and instantiated with $\epsilon$ and $\beta$ values of 0.05

### B.2. Training performance

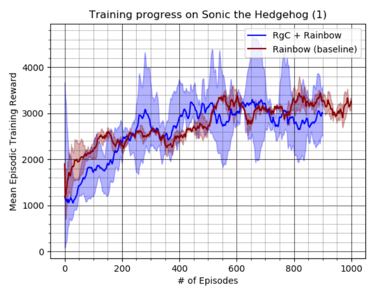

*Figure 2.* Training progress on Sonic the Hedgehog 1 averaged over 5 agents with variance shown in shaded region

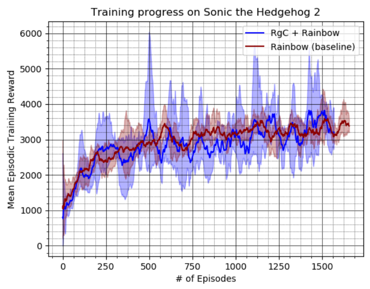

*Figure 3.* Training progress on Sonic the Hedgehog 2 averaged over 5 agents with variance shown in shaded region – Each agent was allowed to complete $2 \times 10^6$ steps in training

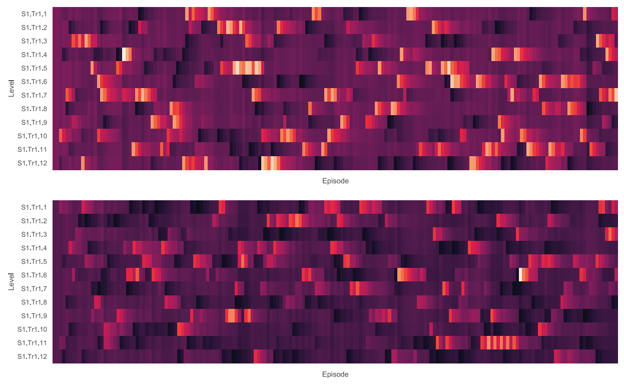

*Figure 4.* Sample heat-maps to visualize how the sampling probabilities over the training distribution evolves during the course of training two independent agents with RgC for Sonic the Hedgehog 1 - train/test split 1. Observe that despite starting with a uniform sampling distribution, the meta-learner eventually begins to focus on specific levels and constantly adjusts its sampling per the needs of the agent being trained. Furthermore, the sampling strategies are different between the two learners, despite operating on the same task distribution, indicating that the meta-learner adapts the the needs of individual learners

## C. Classical Control Tasks

### C.1. Training Environments

We primarily use 3 training environments in all our experiments:

1. Pendulum-v0 from OpenAI Gym (Brockman et al., 2016), modified minimally to allow for programmatic control of the pendulum's mass. State space: $sin\theta$, $\cos\theta$, $\dot{\theta}$, Action space: $Torque$. Sample environment shown in Figure 5
   **Defining success:** Having a maximum deviation of less than 15 degrees from the vertical balancing point over the last 100 steps of a 300-step episode.

2. CartPole-v1 from OpenAI Gym, modified to allow programmatic control of the pole and cart masses, as well as to be treated as a continuous control task, as opposed to one with discrete actions. State space: $x, \dot{x}$ (of cart), $\theta, \dot{\theta}$ of pole. Action Space: $Force$
   **Defining success:** Holding the pole steady (dropping less than 10 degrees) for at least 490 of the 500 total steps of an episode.

3. Custom Unity Ball-pushing task. State space: $x_g, y_g$ position of goal, $x_b, y_b$ position of ball, $\dot{x}_b, \dot{y}_b$ velocity of ball.
   **Defining error:** Distance of the ball from the goal at the time of episode completion.

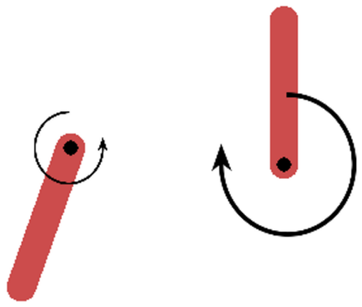

*Figure 5. Sample renders from Gym Pendulum-v0* (Left) Random Initialization, (Right) Successful Completion

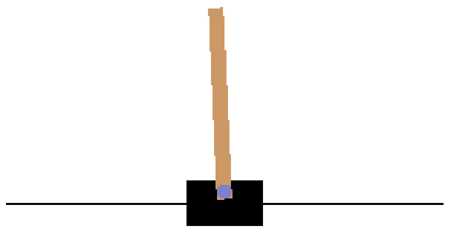

*Figure 6. Sample render from Gym CartPole-v1* Success is determined by maintaining the pole steady

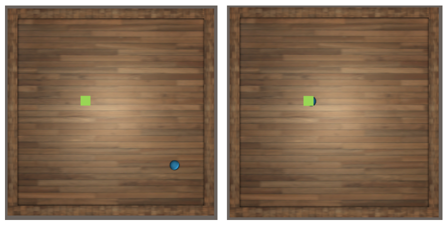

*Figure 7. Sample renders from Unity Ball-pushing* (Left) Random Initialization, (Right) Successful Completion

*Table 4. OpenAI-Gym Pendulum policy success rate (higher is better, darker is worse).* We evaluate multiple policies trained and tested on different OpenAI-Gym Pendulum-v0 environment settings. Rows represent performance for policies trained on a specific mass, columns correspond to specific test masses. Success rate is computed as the fraction of 8 trials with an average maximum deviation of less than 15 degrees, over 6 tests per test mass per trial, from the vertical steady point over the last 100 steps of a 300-step episode.

| TRAIN\TEST | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.75 | 0.63 | 0.50 | 0.25 | 0.38 | 0.38 | 0.38 | 0.25 | 0.13 | 0.13 | 0.38 |
| 4 | 0.75 | 0.88 | 0.75 | 0.75 | 0.75 | 0.63 | 0.50 | 0.38 | 0.38 | 0.13 | 0.59 |
| 6 | 0.88 | 1.00 | 1.00 | 1.00 | 1.00 | 0.88 | 0.88 | 0.75 | 0.38 | 0.25 | 0.80 |
| 8 | 0.75 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 |
| 10 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

*Table 5. Cart-pole success rate for policies trained on individual environment settings (higher is better).* Note that the column and row headings contain the trained/tested pole and cart masses respectively within brackets

| Train/Test | (0.1,1) | (0.1,3) | (0.1,5) | (0.25,1) | (0.25,3) | (0.25,5) | (0.5,1) | (0.5,3) | (0.5,5) | (1,1) | (1,3) | (1,5) | AVG |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (0.1,1) | 1.00 | 0.00 | 0.00 | 0.83 | 0.00 | 0.00 | 0.67 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.21 |
| (0.1,3) | 0.20 | 0.20 | 0.00 | 0.20 | 0.60 | 0.00 | 0.20 | 0.40 | 0.00 | 0.20 | 0.20 | 0.00 | 0.18 |
| (0.1,5) | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.83 | 0.67 | 0.67 | 0.67 | 1.00 | 0.50 | 0.50 | 0.82 |
| (0.25,1) | 0.83 | 0.00 | 0.00 | 0.83 | 0.00 | 0.00 | 0.67 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 |
| (0.25,3) | 0.80 | 0.80 | 0.00 | 0.80 | 0.80 | 0.00 | 0.80 | 0.60 | 0.00 | 0.80 | 0.20 | 0.00 | 0.47 |
| (0.25,5) | 1.00 | 0.75 | 1.00 | 1.00 | 0.75 | 1.00 | 1.00 | 0.75 | 0.50 | 0.75 | 0.75 | 0.50 | 0.81 |
| (0.5,1) | 1.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.83 | 0.00 | 0.00 | 0.17 | 0.00 | 0.00 | 0.25 |
| (0.5,3) | 0.67 | 0.83 | 0.33 | 0.67 | 0.83 | 0.33 | 0.67 | 0.83 | 0.17 | 0.67 | 0.50 | 0.17 | 0.56 |
| (0.5,5) | 1.00 | 0.83 | 0.83 | 1.00 | 0.67 | 0.67 | 1.00 | 0.83 | 0.83 | 1.00 | 0.83 | 0.50 | 0.83 |
| (1,1) | 0.80 | 0.00 | 0.00 | 0.80 | 0.00 | 0.00 | 0.80 | 0.00 | 0.00 | 0.60 | 0.00 | 0.00 | 0.25 |
| (1,3) | 0.83 | 0.83 | 0.17 | 0.83 | 0.83 | 0.17 | 0.83 | 0.83 | 0.17 | 0.83 | 0.67 | 0.17 | 0.60 |
| (1,5) | 1.00 | 0.83 | 0.83 | 0.83 | 0.83 | 0.83 | 1.00 | 0.83 | 0.83 | 0.83 | 0.83 | 0.67 | 0.85 |

## C.2. Experimental Details

DDPG network configuration:

- **Hidden layer configuration**: (400,300)
- **Additional notes**: Code adapted from Patrick Emami's code which is available on Github. Modifications were made to remove the use of tflearn and use only TensorFlow.

Reward-Guided stochastic curriculum parameters:

- $\epsilon$: 0.05 for pendulum and ball-pushing, 0.2 for cart-pole
- $\beta$: 0.05 for pendulum and ball-pushing, 0.2 for cart-pole

## D. Exploring Policy Brittleness in Continuous Control

When training RL policies on the control tasks described in Appendix C, we noted that there was a significant disparity in the performances of trained policies when exposed to different physics settings. For the pendulum task, we observed that policies trained on a heavy pendulum would often generalize well to lighter-weight pendulums, but policies trained on light-weight pendulums would simply fail to work on heavier pendulums, despite having access to and even hitting the same maximum torque. This disparity in performance is captured in Table 4. We observed a similar phenomenon with the Cart-pole environment too (Table 5), where high-mass carts and poles tended to motivate the learning of more robust policies. On the ball-pushing task however, we noted that it was in fact training to manipulate the lightest ball that yielded the best performance (Table 6). This indicated that the problem of brittle policies could not be addressed by just naively training on high-inertia environments.

*Table 6. Unity Ball-pushing median policy error (lower is better).* Performance evaluation of multiple policies trained and tested on different custom Unity ball-pushing environment settings. Rows represent performance for policies trained on a specific ball mass, columns correspond to specific test masses. Errors are computed as the median Euclidean distance of the ball from the goal evaluated on 6 separate trials, with 50 pre-defined tests per trial (to ensure fair comparison between policies).

| Train\Test | 2 | 4 | 6 | 8 | 10 | Avg |
|---|---|---|---|---|---|---|
| 2 | 0.21 | 0.21 | 0.22 | 0.22 | 0.22 | 0.22 |
| 4 | 0.40 | 0.23 | 0.25 | 0.25 | 0.25 | 0.28 |
| 6 | 0.66 | 0.37 | 0.36 | 0.36 | 0.37 | 0.43 |
| 8 | 1.25 | 0.66 | 0.63 | 0.63 | 0.64 | 0.76 |
| 10 | 1.15 | 0.54 | 0.46 | 0.45 | 0.46 | 0.61 |