# Reinforcement Learning Generalization with Surprise Minimization

**Jerry Zikun Chen** [1]

## Abstract

Generalization remains a challenging problem for deep reinforcement learning algorithms, which are often trained and tested on the same set of deterministic game environments. When test environments are unseen and perturbed but the nature of the task remains the same, generalization gaps can arise. In this work, we propose and evaluate a surprise minimizing agent on a generalization benchmark to show an additional reward learned from a simple density model can show robustness in procedurally generated game environments that provide constant source of entropy and stochasticity.

## 1. Introduction

Reinforcement learning (RL) algorithms hold great promises in the pursuit of artificial intelligence as demonstrated by beating human players in board games (Silver et al., 2017; 2018), reaching expert level in video games (Vinyals et al., 2019; Berner et al., 2019), and improving complex robotic control tasks (Haarnoja et al., 2018). However, deep RL agents are often hampered by their struggle to generalize in new environments, even when the semantic nature of the task remain similar (Farebrother et al., 2018; Zhang et al., 2018; Gamrian & Goldberg, 2018). It is evidenced that they are prone to overfitting by memorizing experiences from the training set (Zhang et al., 2018). Therefore, a near-optimal policy can nonetheless produce generalization gaps in unseen game levels (Cobbe et al., 2018; 2019). This problem makes RL agent unreliable for real world applications where robustness is important.

Recent work (SMiRL) (Berseth et al., 2019) inspired by the free energy principle (Friston, 2010) has shown that intelligent behaviours can emerge from minimizing expected future surprises in order to maintain homeostasis. This is motivated by the entropy-increasing nature of the real world that can provide constant novelty without the need for explicit intrinsic motivation to explore. A SMiRL agent alternates between learning a density model that estimates the marginal state distribution under its policy, and improving the policy to seek more predictable stimuli. Probability of an observed state is optimized and predicted by decoding the maintained density model learned from history. Theoretically, this objective of minimizing future surprises in entropic environments tightens a lower bound on the entropy of the marginal state distribution under the policy. Adding a surprise minimizing reward to the raw episodic task reward also shows faster reward acquisition. This framework can potentially help the agent achieve better generalization since it is integral that a surprise minimizing agent learns a robust policy in order to perform well in a dynamically changing environment. When the agent minimizes future surprises throughout training, a bias toward stability and predictability can be learned to counteract the prevailing source of entropy from the environment, which should be present in the test set as well.

To evaluate generalization performance on diverse, and randomized training and test environments, a good candidate is the Procgen benchmark (Cobbe et al., 2019). It consists of 16 game environments designed using procedural content generation (Johnson et al., 2010). Compared to human-generated level layout and progression for each game that are commonly seen in RL game benchmarks, procedural generation creates highly randomized content for each episode that is suitable for evaluating generalization. To do well on the test set, an agent needs to be encouraged to learn a policy that is robust against changes in variables such as level layout, initial location of entities, background designs, et cetera. We show that adding surprise minimizing information through rewards learned by the density model on states of the training history can improve generalization on procedurally generated random game levels that are unseen during training. Since the history consists of episodes in levels that are dynamic and unpredictable, to do well, the agent must learn to stabilize amidst the changing nature of its environment and acquire robust skills through minimizing the chances of seeing surprising states.

[1]Department of Computer Science, University of Toronto, Toronto, Canada. Correspondence to: Jerry Zikun Chen <jzchen@cs.toronto.edu>.

*Proceedings of the $37^{th}$ International Conference on Machine Learning*, Vienna, Austria, PMLR 108, 2020. Workshop on Inductive Biases, Invariances and Generalization in RL (BIG). 

## 2. Related Work

RL benchmarks dealing with generalization have emerged in recent years. Sticky actions (Machado et al., 2018) and random starts (Hausknecht & Stone, 2015) are methods to add stochasticity to the Atari benchmarks. Procedural generation is used to create diverse game environments (Pérez-Liébana et al., 2018; Justesen et al., 2018). In procedurally generated Obstacle Tower (Juliani et al., 2019), the agent is required to learn both low-level control and high-level planning problems. Generalization performance of baseline algorithms like PPO (Schulman et al., 2017) and Rainbow (Hessel et al., 2017) are poor on the Sonic benchmark (Nichol et al., 2018), which is targeted at few-shot learning. Safety Gym (Ray et al., 2019) consists of a suite of 18 high-dimensional continuous control environments for safe exploration under the constrained MDP framework. While their environments do have a high level of randomization, their main objective is to enforce safety and to minimize the constraint cost. Various techniques have been proposed to achieve better generalization for RL agents. It has been shown (Tobin et al., 2017) that robust representations can be learned by the RL agent if diverse observations are provided during training. One example of this involves adding a convolutional layer in between the input image and the neural network policy (Lee et al., 2020). This layer is randomly initialized at every iteration to create a variety of randomized training data. Zhang et al. 2018 report insightful discussions on the nature of RL over-fitting. They experiment on procedurally generated gridworld mazes and find that agents have a tendency to memorize specific levels in a given training set and have risk of overfitting robustly. Experiments on the CoinRun game (Cobbe et al., 2018) investigates the impact of supervised learning regularization techniques including L2 regularization, dropout, data augmentation, and batch normalization, which all help narrow the generalization gap. Similar to Zhang et al. 2018, they also found that injecting stochasticity with $\epsilon$-greedy action selection and increasing entropy bonus in PPO can both improve generalization. Importantly, stochasticity helped generalization to a greater extent than regularization techniques from supervised learning. However, it is believed that forced randomness that is not sourced from the environment can have a negative impact on the learning agent (Hausknecht & Stone, 2015). Rather than altering the action selection procedure, we inject stochasticity through a different channel in the reward by learning state distributions from highly stochastic and dynamic environments in the Procgen benchmark. Since observed states are hard to predict with high entropy in regards to the state marginal distribution, the source of the stochasticity lies inherently from the environment itself.

## 3. Methodology

In Cobbe et al. 2019, agents are trained using PPO (Schulman et al., 2017) for 200M time steps under the hard difficulty. Under this setting, training requires approximately 24 GPU-hours per environment. In our case, for computational efficiency on a single GPU on Colab, we follow the suggestion from Cobbe et al. 2019, both of our training and testing are done under the easy difficulty setting for 25M steps. A finite set of levels is use as the training set and the full distribution of levels are in the test set to evaluate generalization. Similar to Cobbe et al. 2019, we use OpenAI baselines (Dhariwal et al., 2017) implementation of PPO, which has an IMPALA (Espeholt et al., 2018) network as the policy. For clarity, all curves in Section 4 are exponentially smoothed with weight 0.5.

We train and evaluate the PPO baseline on the game CoinRun (Cobbe et al., 2018), the inaugural environment in the Procgen benchmark, used by other works. For further comparison, we also run experiments on another game BossFight, which has a more static observation background (see Figure 1).

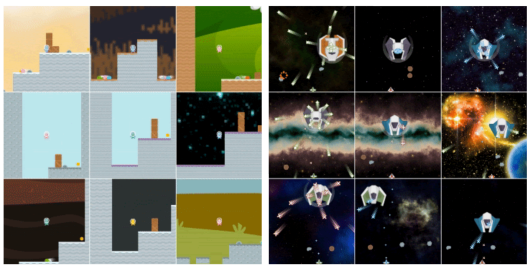

*Figure 1.* CoinRun and BossFight

Following Berseth et al. 2019, we implement the surprise minimizing reward $r_{SM}$ based on a representation density model $p_{\theta_t}(s)$ as shown in Algorithm 1. This is an estimate of the true marginal state distribution under the policy. This model is used to construct the surprise minimizing reward. In the surprise minimization framework, one may think that SMiRL will seek degenerate behaviours that optimize for more familiar states that are high in $p_{\theta_t}(s)$ at the current step. But similar to an agent optimizing for sparse future rewards that might only appear at the end of the episode, the optimal policy does not simply visit states that have high probabilities now, but also those that can update $p_{\theta_t}(s)$ such that it provides high likelihood to the states that it sees in the long term.

### 3.1. Normal Distribution

In Algorithm 1, we first model $p_{\theta_t}(s)$ using independent Gaussian distributions for each dimension in our observa-

**Algorithm 1** PPO + Normal Algorithm

Initialize game environment $Env$
Initialize IMPALA policy $\pi_{\phi_0}$
Initialize Data Buffer $D_0$
**for** $t = 1, \cdots, T$ **do**
  # collect a batched experiences from current policy
  # each batch $\mathbf{b}_t$ consists of multiple episodes
  $\mathbf{b}_t = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t\} \leftarrow Env(\pi_{\phi_{t-1}})$
  $D_t \leftarrow D_{t-1} \cup \{greyscale(\mathbf{s}_t)\}$
  # computed from $D_t$ for
  # each dimensions across states
  $\theta_t = \{\mu_t, \sigma_t\} \leftarrow D_t$
  # Normal SM rewards for each state in $\mathbf{s}_t$
  $\mathbf{r}^t_{SM} = \log p_{\theta_t}(\mathbf{s}_t)$
  $\phi \leftarrow PPO(\phi_t, \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t + \alpha \mathbf{r}^t_{SM}\})$
**end for**

**Algorithm 2** PPO + VAE Algorithm

Initialize game environment $Env$
Initialize IMPALA policy $\pi_{\phi_0}$
Initialize $\text{VAE}_{\psi_0}$
**for** $t = 1, \cdots, T$ **do**
  $\mathbf{b}_t = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t\} \leftarrow Env(\pi_{\phi_{t-1}})$
  # update VAE parameters
  $\psi_t \leftarrow SGD(\{\mathbf{s}_t\})$
  # produce latent representation
  $\mathbf{z}_t \leftarrow \text{VAE}_{\psi_t}(\{\mathbf{s}_t\})$
  # compute parameter of diagonal Gaussian from $\mathbf{z}_t$
  $\theta_t = \{\mu_t, \sigma_t\} \leftarrow \mathbf{z}_t$
  $\mathbf{r}^t_{SM}$ = $\log p_{\theta_t}(\mathbf{z}_t)$ # VAE SM rewards
  $\phi \leftarrow PPO(\phi, \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t + \alpha \mathbf{r}^t_{SM}\})$
**end for**

tions. A buffer is created for storing the most recent batches in the history during training. The size of the buffer is 20 times the mini-batch size. Observations in each batch $s_t$ contains 16384 RGB frame observations with dimensions $64 \times 64 \times 3$. These observations may consist of multiple episodes depending on the agent's performance. We transform and normalize all RGB observations to $64 \times 64$ grayscale images before they are stored in the buffer. Before each mini-batch update to our policy, a surprise minimizing reward is computed from the buffer. Given a particular state $s$ in the buffer $D$, the SM reward is computed as:

$$r_{SM}(s_t) = -\sum_i (\log \sigma_i + \frac{(s_i - \mu_i)^2}{2\sigma_i^2})$$

where $\mu_i$ and $\sigma_i$ are the sample mean and standard deviation of the $i^{th}$ dimension calculated across each state in the data buffer. $s_i$ is value of the $i^{th}$ dimension in $s$. This reward is computed for each batched observations, rather than for individual observations in each episode in SMiRL (Berseth et al., 2019). Therefore, this can be seen as an instance of non-episodic surprise minimization. This is also the case for the VAE method in Algorithm 2.

### 3.2. Variational Autoencoder

Additionally, as presented in Algorithm 2, we take a variational autoencoder (VAE) instead of Normal density to learn a richer representation from observations from batches. We trained the VAE with raw RGB observations without adjustment. Following Berseth et al. 2019, instead of a data buffer, we train this VAE online across all episodes during training since it requires more data. Distinct from the VAE prior $p(\mathbf{s})$, a batch-specific distribution $p_{\theta_t}(\mathbf{z})$ is tracked to compute the SM reward. $p_{\theta_t}(\mathbf{z})$ is represented as a normal distribution with diagonal covariance. Parameters of this distribution are computed from the VAE encoder outputs of the observations in batch $\mathbf{s}_t$ with size $B$ (line 9 in Algorithm 2):

$$z_j = \text{E}[q(z_j|s_j)], \forall s_j \in \mathbf{s}_t, j \in \{1, \cdots, B\}$$

$$\mu = \frac{\sum_{j=0}^{B} z_j}{B+1}, \sigma = \frac{\sum_{j=0}^{B} (\mu - z_j)^2}{B+1}, \theta_t = \{\mu, \sigma\}$$

To compute the SM rewards for each observation, we take the log probability $\log p_{\theta_t}(z_j)$ of this normal distribution evaluated at each $z_j$ in the batch.

In both Normal and VAE cases, the SM reward is added to the raw episodic reward for the specific task via the equation:

$$r_{combined}(s) = r_{task}(s) + \alpha r_{SM}(s)$$

$\alpha$ is a hyper-parameter selected to balance the magnitudes of the two rewards. Generalization in the additional SM reward settings are compared against the PPO baselines. The evaluation score is the mean raw task reward across all episodes for each mini batch update achieved during training and testing. It ranges from 0 to 10 for CoinRun and 0 to 12 for BossFight.

## 4. Experimental Results

### 4.1. Normal Distribution

**CoinRun** In CoinRun, where the agent avoids monsters on a platform to acquire a gold coin, we first evaluate the Procgen PPO baseline algorithm. Figure 2 confirms the finding from Cobbe et al. 2019 that there is a gap between training and testing performances, especially during the last 1.5M steps, where the mean task reward is 7.78 for the training levels, and 9.37 for the testing levels. However, the gap is smaller compared to the one reported in Cobbe et al. 2019 considering the difficulty mode is set to easy.

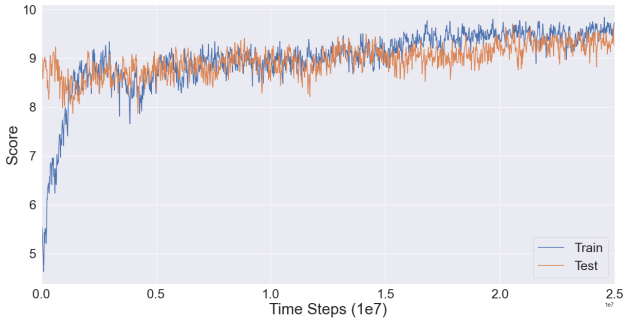


*Figure 2.* CoinRun - Baseline PPO: Train vs. Test Score

The performance of the agent is investigated when it is trained on a combination of both the surprise minimizing reward and the raw task reward from the game. SM reward is defined as the log probability of observed states from normally distributed density model as mentioned in Section 3. The hyper-parameter $\alpha$ of $10^{-4}$ is selected to downscale the SM reward to a similar level as the task reward. In Figure 6, we can see that scores achieved with the combined reward (orange) during 25M steps of training is lower than the baseline (blue). However, on the test set, the agent trained on combined task and Normal SM rewards has comparable scores to the baseline trained on the task reward alone (Figure 7). The comparison in Figure 3 shows that with the combined reward, the task rewards on the test set outperforms the training set at all steps. This suggests a simple Gaussian density model can provide additional signal for robustness by maintaining homeostasis.

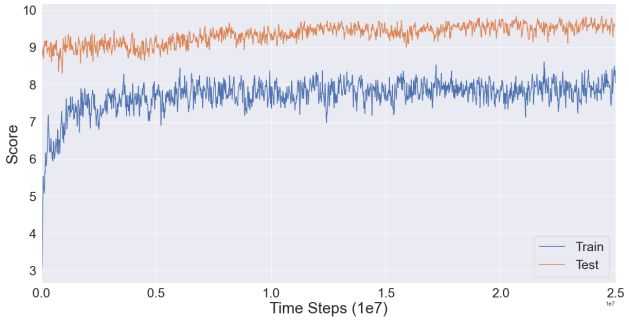


*Figure 3.* CoinRun - PPO + Normal: Train vs. Test Score

**BossFight** Similar to CoinRun, same experiments are produced on BossFight. This game is has a more static visual background. The objective is to remain intact from lasers and destroy the boss star ship. In CoinRun, as the agent is centred at all times, the visual background shift according to how the agent moves. In bossfight, only moving parts in BossFight are the boss, the agent and their lasers. An $\alpha$ of $10^{-6}$ works well to downscale SM rewards. It is found that there is a more prominent gap between train and test curves (Figure 4) in the PPO baseline. Furthermore, the learned policy is slow at attaining decent results on the test set, implying a worse generalization performance than CoinRun. By adding the Normal SM reward, we achieve better task rewards on the test levels (Figure 5 in orange). Comparing test curves (Figure 9), the addition of a NormalSM reward shows a higher task reward throughout testing, indicating its benefit on generalization.

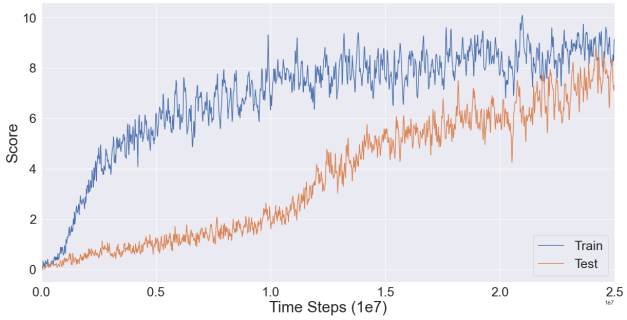


*Figure 4.* Bossfight - Baseline PPO: Train vs. Test Score

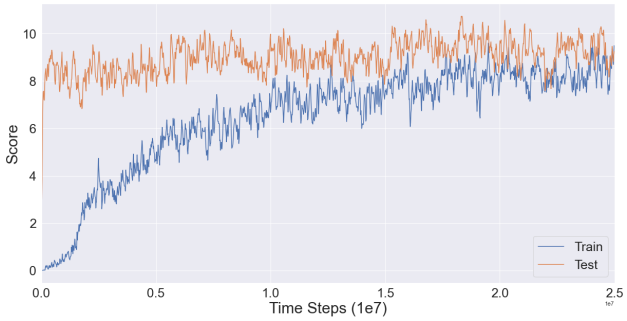


*Figure 5.* Bossfight - PPO + Normal: Train vs. Test Score

### 4.2. Variational Autoencoder

The particular behaviour of a surprise minimizing agent is strongly influenced by the choice of state representation (Berseth et al., 2019). We further implement a variational autoencoder (Kingma & Welling, 2014; Rezende et al., 2014) during training to see if a more complex density model can better improve generalization than the simpler Gaussian case. The VAE has convolutional layers similar to (Kendall et al., 2018), where they utilized it for autonomous driving environments. The autoencoder has a latent dimension of 100. We train this VAE online to produce latent representations over all the states in the batched observations $\mathbf{s}_t$ to produce mean and variance of a multivariate normal distribution with the same dimension as the latent encoding. Every update to the VAE uses the same batch of observations as the policy update. We optimize the VAE across batched

observations to compute the surprise minimizing reward as described in Section 3. Note that due to computational constraints, we train the VAE along with our RL algorithm to 9.3 million steps for both games.

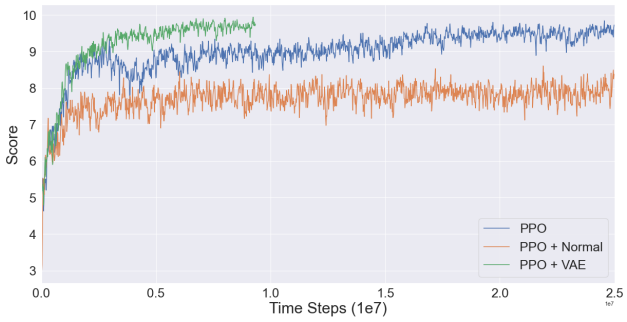


*Figure 6.* CoinRun Train Scores

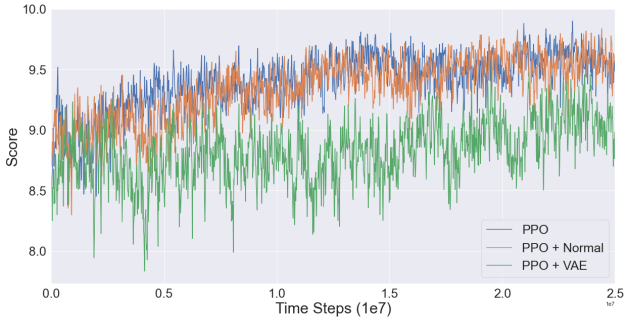


*Figure 7.* CoinRun Test Scores

**CoinRun** We choose $\alpha$ to be $10^{-3}$ for CoinRun. Even though it is stopped early, the task reward for PPO + VAE in Figure 6 (PPO + VAE in green) has already eclipsed the two other methods. SM reward computed from the VAE helps the agent to achieve high task reward significantly faster than the baseline PPO. However, the test results using the policy trained for 9M steps does not achieve better generalization results in Figure 7. We think that the policy in combination with the additional reward from VAE may overfit to the training levels due to its richer representation power and thus the policy cannot generalize as well to the test levels.

**BossFight** For BossFight, $\alpha$ is chosen to be $10^{-5}$ for balancing the two rewards. In the beginning of testing (see Figure 9), it can be observed the same possible overfitting effect of the VAE approach, which results in lower test scores. But as testing progresses, rewards get on par with the PPO + Normal method.

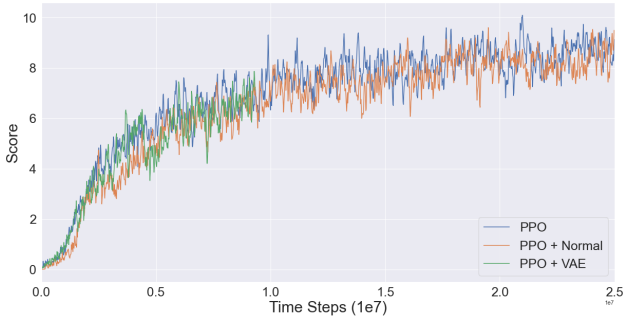


*Figure 8.* BossFight Train Scores

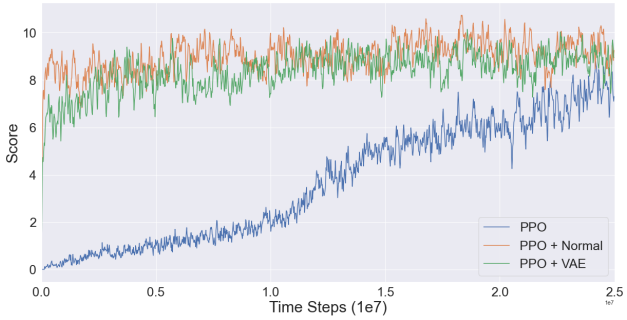


*Figure 9.* BossFight Test Scores

## 5. Discussions

As observed, adding surprise minimizing reward can be helpful for generalization in the context of procedurally generated environments. Surprise minimization aims to predict states by learning a density model through historical trajectories under the policy to minimize future surprises. The assumption underlying this framework is that the environment is constantly changing and the agent needs to maintain stability in order to achieve good performance and avoid surprises. This is a suitable framework for robustness since we would like the agent to be able to generalize under entropic environments, where observations are randomized but the task remains the same. The Procgen benchmark supplies sufficient novel stimulus in the training set by varying layout design, background visuals, locations of entities, and so on. By adding the surprise minimizing reward signal, which is learned from stochastic observations, the agent is encouraged toward stability and predictability that counteracts the prevailing entropy in the environment, thus acquiring more robust skills for future surprises that are present in the test set. Additionally, stochasticity in the Procgen environment is injected through the additional SM reward, which might also encourage better generalization performance.

There are aspects for improvements and further work. First,

our analysis is limited to two games in the benchmark and experiments on other games can help better understand the effectiveness of the two methods. As we have discovered, the additional reward works better in BossFight where the agent tries to avoid lasers and remain intact, this is a different objective than searching for a coin in CoinRun. It is unclear how to properly evaluate how much entropy an environment can provide throughout training and which game environments are more or less entropic. If the degree of how entropic a game can be controlled, experiments can be done to understand the extent to which surprise minimizing reward can help. The choice of the density model can also influence the generalization gap. To narrow the gap and prevent overfitting, careful choice of its structure might be needed for a rich model like the variational autoencoder. Further, the choice of the hyperparameter in determining the magnitude of surprise minimizing reward can make or break the generalization performance. The particular values chosen in the paper might not be optimal afterall. During the process of experimentation, the performance is found to become poor when the surprise minimization reward dominates. So how to balance the SM reward properly specific for the task or the density model can be another topic.

## Acknowledgements

The author thanks Ning Ye for help and discussions, Animesh Garg for the robotics seminar, and the reviewers for helpful commentaries.

## References

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019. URL https://arxiv.org/abs/1912.06680.

Berseth, G., Geng, D., Devin, C., Finn, C., Jayaraman, D., and Levine, S. SMiRL: Surprise Minimizing RL in Dynamic Environments. *CoRR*, arXiv/1912.05510, 2019. URL https://arxiv.org/abs/1912.05510.

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

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588, 2019. URL https://arxiv.org/abs/1912.01588.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. https://github.com/openai/baselines, 2017.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in DQN. *CoRR*, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

Friston, K. J. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11: 127–138, 2010. URL https://www.nature.com/articles/nrn2787.

Gamrian, S. and Goldberg, Y. Transfer learning for related reinforcement learning tasks via image-to-image translation. *CoRR*, abs/1806.07377, 2018. URL http://arxiv.org/abs/1806.07377.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *CoRR*, 2018. URL https://dblp.org/rec/journals/corr/abs-1812-05905.

Hausknecht, M. and Stone, P. The impact of determinism on learning atari 2600 games. In *AAAI Workshop on Learning for General Competency in Video Games*, January 2015. URL http://www.cs.utexas.edu/~pstone/Papers/bib2html/b2hd-AAAI15-hausknecht.html.

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. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

Johnson, L., Yannakakis, G. N., and Togelius, J. Cellular automata for real-time generation of infinite cave levels. 2010. URL https://www.um.edu.mt/library/oar/bitstream/123456789/22895/1/Cellular_automata_for_real-time_generation_of.pdf.

Juliani, A., Khalifa, A., Berges, V., Harper, J., Henry, H., Crespi, A., Togelius, J., and Lange, D. Obstacle tower: A generalization challenge in vision, control, and planning. *CoRR*, abs/1902.01378, 2019. URL http://arxiv.org/abs/1902.01378.

Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729, 2018. URL http://arxiv.org/abs/1806.10729.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., and Shah, A. Learning to drive in a day. *CoRR*, abs/1807.00412, 2018. URL http://arxiv.org/abs/1807.00412.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014. URL https://arxiv.org/abs/1312.6114.

Lee, K., Lee, K., Shin, J., and Lee, H. Network randomization: A simple technique for generalization in deep reinforcement learning. In *ICLR*, 2020. URL https://openreview.net/forum?id=HJgcvJBFvB.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61, 2018. URL https://jair.org/index.php/jair/article/view/11182.

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. URL http://arxiv.org/abs/1804.03720.

Pérez-Liébana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., and Lucas, S. M. General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms. *CoRR*, abs/1802.10363, 2018. URL http://arxiv.org/abs/1802.10363.

Ray, A., Achiam, J., and Amodei, D. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019. URL https://arxiv.org/abs/1910.01708.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *International Conference on Machine Learning*, 32, 2014. URL https://arxiv.org/abs/1401.4082.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Aja Huang, A. G., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *Nature*, 550, 2017. URL https://www.nature.com/articles/nature24270.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou1, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362, 2018. URL https://science.sciencemag.org/content/362/6419/1140.

Tobin, J., Fong, R. H., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017. URL https://ieeexplore.ieee.org/document/8202133.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., and et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 2019. URL https://www.nature.com/articles/s41586-019-1724-z.

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