Syllabus: Curriculum Learning Made Easy

Anonymous Author(s)
Affiliation
Address
email

Abstract

Curriculum learning has been a quiet yet crucial component of many of the high-profile successes of reinforcement learning. Despite this, none of the major reinforcement learning libraries support curriculum learning or include curriculum learning algorithms. Curriculum learning methods can provide general and complementary improvements to RL algorithms, but they often require significant, complex changes to agent training code. We introduce Syllabus, a library for training RL agents with curriculum learning, as a solution to this problem. Syllabus provides a universal API for implementing curriculum learning algorithms, a collection of implementations of popular curriculum learning methods, and infrastructure for easily integrating them into existing distributed RL code. Syllabus provides a clean API for each of the complex components of these methods, dramatically simplifying the process for designing new algorithms or applying existing algorithms to new environments. Syllabus also manages the multiprocessing communication required for curriculum learning, alleviating one of the key practical challenges of using these algorithms. We hope Syllabus will improve the process of developing and applying curriculum learning algorithms, and encourage widespread adaptation of curriculum learning.

1 Introduction

Curricula have been a core component of many of the successes of reinforcement learning. AlphaGo [Silver et al., 2016] was trained with self-play, AlphaStar used a novel league training method to achieve grandmaster level play in Starcraft II [Vinyals et al., 2019], and GT Sophy [Wurman et al., 2022] was recently trained to outrace professionals in Gran Turismo with a manually designed curriculum. Curriculum learning methods, both automatic and manual, fit naturally into the RL framework by modifying the distribution of states that an agent experiences. Both in theory and in practice, these methods often provide orthogonal benefits to reinforcement learning algorithms and can be implemented on top of most RL algorithms with minimal restrictions. Despite the near ubiquity of curriculum learning in successful applications of reinforcement learning, there is relatively little support for these methods in standard RL tools, largely due the complexity of curriculum learning code. Curriculum learning methods may modify many parts of the training process, such as the environment initialization and reward functions, and might even introduce new neural networks to train. Automatic curriculum learning methods may update their internal state based on information from the training process or from the environment which typically run in separate processes. In practice, this entanglement of curriculum learning and RL code makes it challenging to take curriculum learning source code and apply it to a new project with pre-existing RL code.

Syllabus addresses this challenge in a number of ways. Even the task of structuring the software architecture for curriculum learning can be daunting. Syllabus clearly designates responsibilities to a separate Curriculum class, and implements changes to the environment through a Task Wrapper,

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

limiting curriculum code to two locations in the code base. Within the Task Wrapper, users can also define a Task Space for their environment that represents the full space of tasks for an environment. These APIs are described in more detail in section 4. We validate our API by implementing several popular curriculum learning methods, demonstrating that it is sufficiently general. Some methods like league training [Vinyals et al., 2019] may be outside the range of Syllabus's Curriculum API, but Syllabus makes it easy to combine one of its methods with your own custom curriculum code.

2 Background

Curriculum Learning has been studied in the context of deep learning for many years [Bengio et al., 2009, Elman, 1993]. More recently, it has been used to improve the performance of reinforcement learning agents. Curriculum learning encompasses a wide range of methods targeted at changing the distribution of data used to train an agent. The goal is to increase the final performance or sample efficiency of RL agents on a single environment or range of tasks by sampling tasks that provide the maximum learning value. Narvekar et al. [2020] presents a more thorough taxonomy and survey of existing curriculum learning methods.

Many diverse methods fall under the broad definition of curriculum learning. Exploration bonuses like curiosity [Pathak et al., 2017] or [Bellemare et al., 2016, Taiga et al., 2021, Henaff et al., 2022] induce a curriculum by incentivizing the agent to explore unseen section of the state space. Methods like self-play or league-play [Vinyals et al., 2019] create an implicit curriculum by training the opponents in a multiplayer game [Leibo et al., 2019]. Progressively more capable opponents lead to progressively more difficult tasks for an agent. Most task-based methods follow the general rule of proposing tasks that are hard yet solvable or tasks which the agent is recently performing well on [Graves et al., 2017, Kanitscheider et al., 2021]. In general, curriculum methods try to create a sequence of increasingly complex or difficult tasks.

Unsupervised Environment Design (UED) is another paradigm for curriculum learning proposed by Dennis et al. [2020]. They differentiate UED as a framework in which environments have unspecified parameters, thereby forming an underspecified MDP. Those parameters are used to produce a distribution of solvable tasks, and the goal of a UED method is to produce a policy that generalizes across a large set of tasks.

3 Design Philosophy

Syllabus was designed to solve a number of challenges in automatic curriculum learning. Our main goal is to simplify the process of testing and developing new curriculum learning methods. Syllabus is also built to manage the multiprocessing communication required for curriculum learning, which can be challenging to combine with existing distributed reinforcement learning.

These objectives motivated the following key points of our design philosophy:

1. Syllabus should be agnostic to the choice of reinforcement learning framework.
2. Syllabus should be as general as possible to support many popular curriculum learning methods.
3. Using Syllabus should require minimal changes to existing RL code.
4. If a method requires more complex code, complexity of the code should scale with the complexity of the curriculum learning method. Simple methods should remain simple to use.
5. Single-file implementations of individual algorithms.

The first point motivates many of the implementation choices in Syllabus which may seem odd in isolation. It requires that we honor the Gym [Brockman et al., 2016] environment API and write systems that work well with the many different forms of multiprocessing used throughout reinforcement learning libraries.

The second, third, and fourth point pertain to the Curriculum API that Syllabus defines. These goals often conflict due to the wide range of methods that need to be supported under a single API. Some curricula may require metrics from the training process while others only utilize rewards

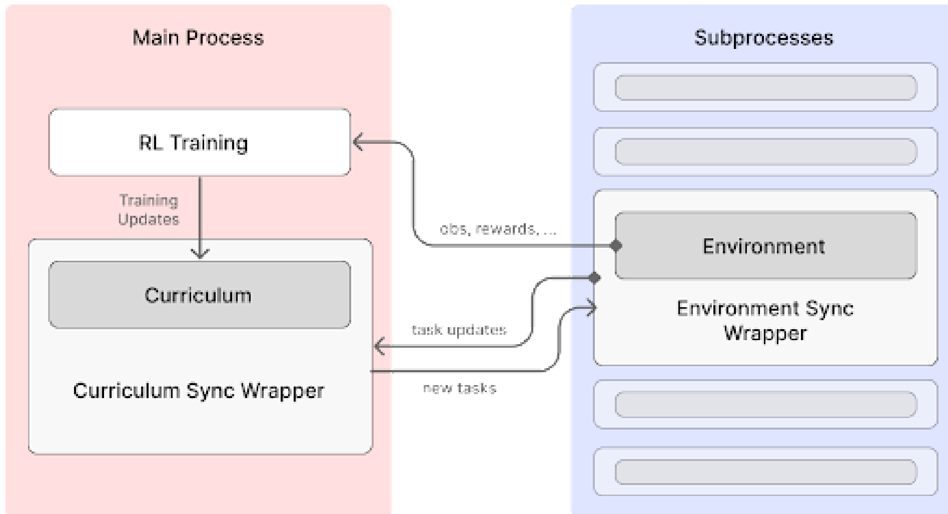

Figure 1: Syllabus with a standard distributed RL training setup.

from the environments. These components typically run in separate processes and therefore require very different interfaces. In line with the fourth design goal, when one method requires additional changes to the training code, we prefer to have a heterogenous API for different methods rather than complicate the API for all methods. To reduce confusion, we thoroughly document these changes and raise warnings for common user errors resulting from those differences. We explain our approach to multiprocessing and balancing these design challenges more in subsection 5.2.

Finally, we choose to use single-file or few-file implementations of curriculum methods inspired by the success of CleanRL [Huang et al., 2022], a popular RL library which strictly uses single-file implementations to simplify research engineering.

4 Syllabus APIs

Syllabus defines APIs for the key components of curriculum learning; the Curriculum API, the Task Space API, and the Task Wrapper API. These are the main components that users will need to modify or interact with to develop or apply curriculum learning methods, so they are each designed to be both simple and general enough to support future use cases.

4.1 Curriculum API

In Syllabus, a Curriculum is responsible for maintaining a distribution over the task space and implementing a sampling function for selecting tasks. Automatic curriculum learning methods require feedback from the RL process to update their sampling distribution. Syllabus provides multiple optional methods which a Curriculum can implement to receive updates from different sources. It can either be updated directly by the main training process, or it can automatically receive updates from the environments through Syllabus's multiprocessing infrastructure. The main components of the API are shown in Figure 2.

4.2 Task Space API

Most curriculum learning methods operate over a distribution of tasks. The task space API defines the range and bounds of this distribution, as well as a conversion between a simple task identifier and the full task definition expected by the environment. This simple identifier can be used to index the task within the task space which allows us to define a subset of validation tasks.

One of the design challenges of applying curriculum learning to a new domain is defining the task space of the environment. In most benchmark environments (like Procgen [Cobbe et al., 2020] or Minigrid [Chevalier-Boisvert et al., 2023]) task spaces are low-dimensional discrete or continuous

Figure 2: Syllabus's Curriculum API.

```python
class Curriculum:
    """API for defining curricula to interface with Gym environments."""

    @property
    def num_tasks(self) -> int:
        """Counts the number of tasks in the task space, if countable."""

    @property
    def tasks(self) -> List[tuple]:
        """ List all of the tasks in the task space, if enumerable."""

    def update_task_progress(
        self, task: Any, progress: Tuple[float, bool]
    ) -> None:
        """ Update the curriculum with a task and its progress.
            Progress is defined by the environment's TaskWrapper. """

    def update_on_step(
            self, obs: Any, rew: float, done: bool, info: dict
        ) -> None:
        """ Update the curriculum with the environment outputs
            for the most recent step. """

    def update_on_demand(self, metrics: Dict):
        """ Update the curriculum with arbitrary inputs.
            Typically used to incorporate gradient or error-based
            metrics from the training process. """

    def _sample_distribution(self) -> List[float]:
        """ Returns a sample distribution over the task space.
            Any curriculum that maintains a true probability distribution
            should implement this method to retrieve the distribution. """

    def sample(self, k: int = 1) -> List:
        """ Sample k tasks from the curriculum. """
```

spaces. In many real environments the space of possible tasks might be a combination of discrete and continuous variables, or a complex predicate system such as in Neural MMO [Suarez et al., 2019]. Curriculum learning algorithms typically only support one specific task space representation. The Task Space API is designed to alleviate some of the challenges of defining task spaces and identifying which curriculum methods can be used with a particular task space. It automatically encodes tasks into a Gym Space which defines the full range of tasks.

4.3 Task Wrapper API

Outside of benchmark environments, the user might need to write code to reinitialize an environment for each task. Syllabus provides a Task Wrapper API to facilitate this configuration process. The main change that this introduces to the standard Gym API is adding a 'task' property to the environment and a 'new_task' argument to the environment's 'reset()' function. This allows the user to optionally assign a new task to the environment while resetting the environment. If necessary, this API can also be used to add a task system to an environment that does not natively support one. For example, in Figure 3 we use this API to add a simple task system to CartPole which changes the range from which the initial location of the cart is sampled. Finally, the task wrapper can also define a progress metric, a float value in the range [0.0, 1.0] that defines how complete the current task is. In the simplest case, this can be a binary value that is 1.0 when the task is completed and 0.0 while it is not. This is allows Syllabus to support learning-progress metrics like the one proposed in Kanitscheider et al. [2021].

5 Implementation

5.1 Implemented Curriculum Learning Methods

Syllabus provides single-file implementations of popular curriculum learning methods including Domain Randomization, Prioritized Level Replay [Jiang et al., 2021a] and the learning progress curriculum proposed by Kanitscheider et al. [2021], with plans to add many more. Each of these methods are tested with Syllabus's infrastructure and include warnings for common user errors.

5.2 Multiprocessing Infrastructure

Syllabus's multiprocessing infrastructure uses a bidirectional sender-receiver model in which the curriculum sends tasks and receives environment outputs, while the environment receives tasks and sends outputs. Environments run the provided task in the next episode and the curriculum can use the outputs to update its task distribution. This communication layer is implemented in two wrappers that add functionality while maintain the same interface. The curriculum synchronization wrapper adds multiprocessing functionality to a Curriculum and an environment synchronization wrapper, similar to the environment wrappers used extensively in reinforcement learning code, adds the same functionality to the environment. You can also call the curriculum directly from the main learner process to update it with training metrics. Crucially, adding Syllabus's functionality to existing RL training code requires only a few lines of code. Figure 1 shows a diagram of how these components interconnect. These synchronization wrappers automatically send the curriculum environment outputs at each step and episodic updates on task progress. All updates are batched to reduce multiprocessing overhead, and the per-step updates can be disabled to improve performance if they are not used by the chosen curriculum learning method.

Most of the user facing Curriculum and environment code follows our design goal of single-file implementations, while the multiprocessing infrastructure is more engineered to ensure stability and reduce the risk of bugs. To guarantee that researchers will not need to spend time reading or debugging this code, Syllabus include thorough integration tests, smoke tests, regression tests, and optimization benchmarks for the entire multiprocessing infrastructure, tested with all of the implemented curriculum learning methods.

5.3 Integration

Following our goal of requiring minimal code changes, the example in Figure 3 shows how easy it is to add a simulated annealing curriculum to RLLib training code. In addition, because each method is implemented with the same API, replacing this curriculum with another only requires a single change. Simply initialize a different curriculum (line 28 in Figure 3), and it will work as expected for most methods. For the few methods that require additional changes, warnings will be raised if those changes are not made. For example, if Prioritized Level Replay does not receive any TD errors from the training process after a full batch of environment experience, it will raise and error and direct users to documentation for adding the missing code.

5.4 Optimization

As a consequence of the choice to use a separate multiprocessing system from the RL training loop, Syllabus incurs some unavoidable computational costs. Specifically, receiving and sending information in the environments decreases the effective steps per second of each environment, while sampling and sending tasks in the actor process increases the computational load on the main process. We perform experiments on the NetHack Learning Environment [Küttler et al., 2020] to demonstrate the effect of this choice on overall steps per second. We evaluate with Domain Randomization, a computationally lightweight method, to isolate the impact of our multiprocessing infrastructure. The results are show in Table 1 for experiments run with 128 environments and 50 episodes on a 32 core Intel i9-13950HX. We test Syllabus using both Python's native multiprocessing package and Ray as the backend. Syllabus allows you to disable per-step updates for ACL methods that do not require

Figure 3: Adding curriculum learning with Syllabus to RLLib training code with just a few lines of code.

```
import gym
from ray.tune.registry import register_env
from ray import tune
from gym.spaces import Box
from .task_wrappers import CartPoleTaskWrapper

+from syllabus.core import RaySyncWrapper, make_ray_curriculum
+from syllabus.curricula import SimpleBoxCurriculum
+from syllabus.task_space import TaskSpace

if __name__ == "__main__":
+   # Define a task space
+   task_space = TaskSpace(Box(-0.3, 0.3, shape=(2,)), [])

    def env_creator(config):
        env = gym.make("CartPole-v1")
        # Wrap the environment to change tasks on reset()
        env = CartPoleTaskWrapper(env)
+       # Add environment sync wrapper
+       env = RaySyncWrapper(
+           env, default_task=(-0.02, 0.02), task_space=task_space
+       )
        return env

    register_env("task_cartpole", env_creator)

+   # Create the curriculum
+   curriculum = SimpleBoxCurriculum(task_space)
+   # Add the curriculum sync wrapper
+   curriculum = make_ray_curriculum(curriculum)

    config = {
        "env": "task_cartpole",
        "num_gpus": 1,
        "num_workers": 8,
        "framework": "torch",
    }

    tuner = tune.Tuner("APEX", param_space=config)
    results = tuner.fit()

```

them, instead only sending updates and new tasks at the end of each episode. We show results for both of these scenarios. Note that the NLE is an extremely fast environment, and often RL training with larger architectures is bottle-necked by policy optimization rather than environment iteration time. We expect Syllabus's impact on performance (as a percentage of total computation) to be much lower for more computationally intensive environments and when combined with RL training.

5.5 Implemented Curriculum Learning Methods

Curriculum learning algorithms vary as much as RL algorithms in complexity. The simplest methods like Domain Randomization never update their sampling distributions and have simple logic for sampling tasks. The most complex methods, such as league training, might need to train entire new agents and maintain many sets of network weights. Syllabus aims to eventually support all

Table 1: Syllabus Performance Costs

Multiprocessing	NLE	Syllabus (Episodic Updates)	Syllabus (Step Updates)
Native Python	58.5s	63.0s (+7.7%)	93.0s (+59.0%)
Ray	71.12s (100%)	84.7s (+19.1%)	134.2s (+88.7%)

curriculum learning paradigms, but it is currently limited to a subset of paradigms that minimally alter the training code. Table x shows some of the key implementation requirements of curriculum learning methods and how they correspond to some popular methods.

6 Related Work

The closest work to Syllabus is RLLib [Liang et al., 2018], which is a large library of reinforcement learning algorithms. RLLib allows you to set the task of an environment, which is similar to Syllabus's Task Environment API, and it manages all of the multiprocessing through Ray. RLLib does not implement any curriculum learning methods or provide utilities for updating a curriculum using metrics from the environment or training process, though it would be possible to implement these through a combination of callbacks and Ray Actors.

Another related library is the Dual Curriculum Design library [Jiang et al., 2022], which incorporates multiple Unsupervised Environment Design Methods in a single repository. It includes implementations of PLR [Jiang et al., 2021a], PAIRED [Dennis et al., 2020, Mediratta et al., 2023], Robust PLR, REPAIRED [Jiang et al., 2021b], and ACCEL [Parker-Holder et al., 2022]. Unlike Syllabus, which provides tools for adding curriculum learning to existing RL code, the DCD library encourages users to iterate on top of it by adding new RL algorithms or environments.

7 Conclusion

Curriculum learning is a sub-field of reinforcement learning that is under-utilized and under-researched relative to its impact on well-known RL applications and empirical benefits. We believe that this is largely due to the practical and engineering challenges associated with these methods. Syllabus is open-sourced on GitHub with a complete documentation website, and can be installed as a pip package. We are continuing to actively develop Syllabus and hope that it will help to encourage the more widespread use of curriculum learning.

References

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016. doi: 10.1038/nature16961.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019. URL `https://api.semanticscholar.org/CorpusID:204972004`.

Peter Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael Thomure, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602:223–228, 02 2022. doi: 10.1038/s41586-021-04357-7.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL `https://doi.org/10.1145/1553374.1553380`.

Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99, 1993. ISSN 0010-0277. doi: https://doi.org/10.1016/0010-0277(93)90058-4. URL `https://www.sciencedirect.com/science/article/pii/0010027793900584`.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *arXiv preprint arXiv:2109.11052*, 2021.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

Joel Z Leibo, Edward Hughes, Marc Lanctot, and Thore Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv preprint arXiv:1903.00742*, 2019.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. Pmlr, 2017.

Ingmar Kanitscheider, Joost Huizinga, David Farhi, William Hebgen Guss, Brandon Houghton, Raul Sampedro, Peter Zhokhov, Bowen Baker, Adrien Ecoffet, Jie Tang, et al. Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft. *arXiv preprint arXiv:2106.14876*, 2021.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Shengyi Huang, Rousslan Fernand JulienDossa Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João GM Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *The Journal of Machine Learning Research*, 23(1):12585–12602, 2022.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021a.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062. PMLR, 2018.

Minqi Jiang, Ishita Mediratta, Mikayel Samvelyan, and Jayden Teoh. Dual curriculum design. `https://github.com/facebookresearch/dcd`, 2022.

Ishita Mediratta, Minqi Jiang, Jack Parker-Holder, Michael Dennis, Eugene Vinitsky, and Tim Rocktäschel. Stabilizing unsupervised environment design with a learned adversary. *arXiv preprint arXiv:2308.10797*, 2023.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021b.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.