Parallel Curriculum Experience Replay in Distributed Reinforcement Learning

Yuyu Li
University of Science and Technology of China
Hefei, China
yuyuli@mail.ustc.edu.cn

Jianmin Ji
University of Science and Technology of China
Hefei, China
jianmin@ustc.edu.cn

ABSTRACT

Distributed training architectures have been shown to be effective to improve the performance of reinforcement learning algorithms. However, their performances are still poor for problems with sparse rewards, e.g., the scoring task with or without goalkeeper for robots in RoboCup soccer. It is challenging to solve these tasks in reinforcement learning, especially for those that require combining high-level actions with flexible control. To address these challenges, we introduce a distributed training framework with parallel curriculum experience replay that can collect different experiences in parallel and then automatically identify the difficulty of these subtasks. Experiments on the domain of simulated RoboCup soccer show that, the approach is effective and outperforms existing reinforcement learning methods.

KEYWORDS

Distributed Training; Reinforcement Learning; Curriculum Learning

ACM Reference Format:
Yuyu Li and Jianmin Ji. 2021. Parallel Curriculum Experience Replay in Distributed Reinforcement Learning. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 8 pages.

1 INTRODUCTION

Distributed training architectures, that separate learning from acting and collect experiences from multiple actors running in parallel on separate environment instances, have become an important tool for deep reinforcement learning (DRL) algorithms to improve the performance and reduce the training time [5, 10, 18, 20, 30]. For instance, they have been applied to playing the game of GO [40], the real-time strategy game of StarCraft II [42, 43], and the multiplayer online battle arena game of Dota 2 [33].

However, most existing distributed training architectures share two major limitations. In specific, they assume that their multiple actors only interact with the same environment, which limits their ability to collect experiences from different environments. They focus on collecting experiences for agents working on the original task and improving their performance on that task, which limits their ability to learn useful knowledge from subtasks to speed up the training. These limitations partially explain the decreased performance of distributed training architectures on problems with sparse rewards, like goal-oriented tasks. In particular, goal-oriented tasks require an agent to manipulate objects into a desired configuration, which are common in the robotics domain [38], e.g., the scoring task with or without goalkeeper for robots in RoboCup soccer [22].

On the other hand, curriculum learning [6] has been considered as a useful tool for DRL to accelerate the learning process for a sparse-reward problem, e.g., a goal-oriented task, by splitting the task into a sequence of progressively more difficult subtasks. In particular, [11] proposed a method named "reverse curriculum", which generates a curriculum that allows the robot to gradually learn to reach the goal from a set of starting positions increasingly far away from the goal. The assumption behind the method is that, it is easier for the task starting from a far-away position if the robot had learned how to achieve the goal starting from nearby positions. For example, the scoring task without a goalkeeper follows the assumption. However, the assumption may not hold when there was a goalkeeper, while shooting strategies are quite different for different starting positions and the robot does not have to move to nearby positions to shoot and score. Moreover, to avoid the catastrophic forgetting problem [27], "reverse curriculum" needs to gradually collect experiences from the easy-to-hard sequence of tasks and incrementally expand the experience reply buffer for harder tasks. This process can seriously reduce the learning efficiency in some cases, which will be illustrated in our experiments.

In this paper, we address the above issues by combining curriculum learning and distributed reinforcement learning. We show that the parallel training of the robot with tasks in the curriculum can improve the performance. In the scoring task with a goalkeeper, it is observed that experiences obtained from a far-away starting position can also help the robot to learn proper strategies for nearby positions. Then training the robot following the reversion of the curriculum, i.e., following the hard-to-easy task sequence, can also improve the performance in some cases. Moreover, the parallel training can effectively mitigate the catastrophic forgetting problem and improve the learning efficiency for hard tasks.

Based on the distributed training framework, by combining parallel collecting experiences from tasks in the curriculum, we propose Distributed Parallel Curriculum Experience Replay (DPCER), a distributed system that can train multiple tasks with different levels of difficulty at the same time and transfer the knowledge learned from simple tasks to difficult tasks. Experiments on the domain of simulated RoboCup soccer show that, the approach is effective and outperforms existing reinforcement learning algorithms.

The main contributions of the paper are:

Jianmin Ji is the corresponding author.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

- We show that the parallel training of the robot with tasks in the curriculum can improve the performance of DRL algorithms.
- We introduce Distributed Parallel Curriculum Experience Replay (DPCER), a distributed DRL paradigm with parallel curriculum experience replay for goal-oriented tasks.
- We implement a DRL algorithm under DPCER for the Half Field Offense (HFO) task [15] in the domain of simulated RoboCup soccer.
  - Experiments show that the algorithm is effective and outperforms existing DRL algorithms in tests of HFO.
  - To the best of our knowledge, this algorithm is the first algorithm that succeeds in applying distributed DRL in discrete-continuous (parameterized) hybrid action space [25].

2 RELATED WORK

2.1 Distributed Training in DRL

To yield more impressive results, a general approach in deep learning is to use more computational resources [8] and work with larger datasets [9]. Recently, distributed learning systems have been applied for deep reinforcement learning. Prior approaches have relied on asynchronous SGD-style learning (e.g., A3C [28] and ADPG-R [35]), batched data collection for high GPU throughput (e.g., GA3C [4], BatchPPO [14]), and more recently, multiple CPU actors for experience generation and a single GPU learner for model update (e.g., Ape-X [18], and IMPALA [10]). Our method is different from these asynchronous gradient methods, as they share gradients between decentralized learners while we collect experiences for the learner. Recently, Ape-X and it's extensions, i.e., D4PG [5], R2D2 [20], Agent57 [37], have shown state-of-the-art results in several benchmarks. Inspired by these methods, our method, DPCER, also separates the experience generation from the centralized learning. Different from these methods, DPCER can train multiple tasks with different levels of difficulty at the same time and transfer the knowledge learned from simple tasks to difficult tasks.

2.2 Curriculum Learning in DRL

Curriculum learning has been applied to train reinforcement learning agents for a long time. A recent result is to speed up the training in robotics domains [24], computer games [44], and Half Field Offense [32], which rely on manually designed curricula. An attempt to automatically construct curricula is to choosing proper tasks in the sequence of curriculums during the training process, which can be considered as a multi-armed bandit problem. The idea has been successfully applied to supervised sequence learning tasks [13], discrete sets of environments [26], and continuously parameterized environments [36]. [2] proposes a "general curriculum" to train a robot to shoot a ball into a goal based on vision inputs. The idea is to create a series of tasks, where the agent's initial state distribution starts close to the goal state, and is progressively moved farther away in subsequent tasks, inducing a curriculum of tasks. [11] proposes a method named "reverse curriculum", which generates a curriculum that allows the robot to gradually learn to reach the goal from a set of starting positions increasingly far away from the goal. The assumption behind the method is that, it is easier for the task starting from a far-away position if the robot had learned how to achieve the goal starting from nearby positions. Both of above methods assume that the goal state is known, while our work focuses on training all tasks parallelly and controlling the learning process automatically. Moreover, our approach is implemented in a distributed training paradigm.

2.3 Parameterized Action Space in DRL

On the other hand, parameterized actions in DRL [25] are composed of discrete actions with continuous action-parameters, which are very common in computer games and robotics. In computer games, parametrized actions DRL has been applied in King of Glory (KOG) [45], a popular mobile multi-player online battle arena game, and Ghost Story [12], a fantasy massive multi-player online role-playing game. In robotics, parameterized actions are also involved in simulated human-robot interaction [21] and terrain-adaptive bipedal and quadrupedal locomotion [34].

Half Field Offense (HFO), a subtask in the domain of simulated RoboCup soccer, that a set of offensive agents attempt to score on a set of defensive agents, is becoming the de facto standard platform for evaluating various parameterized action DRL algorithms [7, 16, 45]. However, those algorithms require reward shaping to handle Test 1v0, i.e., the scoring task without goalkeeper, and are unable to solve Test 1v1, i.e., the scoring task with goalkeeper.

3 PRELIMINARIES

Parameterized actions in DRL [25] are composed of discrete actions with continuous action-parameters, which provides a framework for solving complex domains that require combining high-level actions with flexible control. Notice that, HFO in the RoboCup 2D soccer, i.e., the experimental environment in the paper, is an MDP problem with parameterized actions. In HFO, a set of offensive agents attempt to score on a set of defensive agents. When an agent chooses an action type 'Move', it needs to specify its continuous parameters for the indicated direction with a scalar power.

Before we delve into the model, we first present a mathematical formulation of Parameterized Action MDPs (PAMDPs) along with a DRL algorithm in RoboCup 2D. Then we briefly review architectures that use distributed training to collect replay episodes in DRL. At last, we specify the definition of curriculum learning in RL and goal-oriented tasks.

3.1 Parameterized Action MPDs

PAMDPs is a special class of MDPs where the state space is continuous, $\mathcal{S} \subseteq \mathbb{R}^n$, and the action space is defined as the following parameterized structure:

- $\mathcal{A}_d = \{1, \ldots, K\}$ is a finite set of discrete actions,
- for each discrete action $k \in \mathcal{A}_d$, $\mathcal{X}_k \subseteq \mathbb{R}^{m_k}$ is a set of continuous action-parameters with dimensionality m_k,
- (k, x_k) is an action, where $k \in \mathcal{A}_d$ and $x_k \in \mathcal{X}_k$.

Then the action space is given by

$$\mathcal{A} = \bigcup_{k \in \mathcal{A}_d} \{ (k, x_k) \mid x_k \in \mathcal{X}_k \},$$

which is the union of each discrete action with all possible action-parameters for that action.

A Parameterized Action Markov Decision Process (PAMDP) [25] is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the set of all states, $\mathcal{A}$ is the parameterized action space, $P(s' \mid s, k, x_k)$ is the Markov state transition probability function, $R(s, k, x_k, s')$ is the reward function, and $\gamma \in [0, 1)$ is the future reward discount factor. An action policy $\pi : \mathcal{S} \to \mathcal{A}$ aims to maximize the the expected discounted return following the current policy thereafter.

3.2 Multi-Pass Deep Q-Networks

Multi-Pass Deep Q-Networks (MP-DQN) [7] combines DQN [29] and DDPG [23] to handle parameterized actions. Given a PAMDP problem, MP-DQN first applies an action-parameter choosing network with parameters θ_X to map a state to a vector of continuous action-parameters for discrete actions, i.e.,

$$X(\cdot; \theta_X) : \mathcal{S} \to (\mathcal{X}_1, \ldots, \mathcal{X}_K).$$

We use $X_k(s; \theta_X)$ to denote the k'th element in the resulting vector, i.e., the action-parameter in $\mathcal{X}_k$ for the discrete action k given the state s.

Then MP-DQN uses a Q-network with parameters θ_Q to approximate the action-value function, i.e.,

$$Q(\cdot; \theta_Q) : (\mathcal{S} \times \mathcal{X}\mathbf{e}_1, \ldots, \mathcal{S} \times \mathcal{X}\mathbf{e}_K) \to \mathbb{R}^K \times \mathbb{R}^K,$$

where $\mathcal{X}\mathbf{e}_k$ is the set of vectors of the form $\mathbf{xe}_k = (0, \ldots, 0, x_k, 0, \ldots, 0)$, $x_k \in \mathcal{X}_k$, which is the joint action-parameter vector where each x_j, $j \neq k$ is set to zero. This causes all "false gradients" to be zero, i.e., $\frac{\partial Q_k}{\partial x_j} = 0$ when $j \neq k$, and completely negates the impact of the network weights for unassociated action-parameters x_j from the input layer, making Q_k only depend on x_k.

The output of the Q-network is the following matrix:

$$\begin{pmatrix} \mathbf{Q_{11}} & \cdots & Q_{1K} \\ \vdots & \ddots & \vdots \\ Q_{K1} & \cdots & \mathbf{Q_{KK}} \end{pmatrix},$$

where Q_{ik} is the Q-value for the discrete action k generated on the i'th pass where x_i is non-zero. Only the diagonal elements Q_{kk} are valid and used in the final output.

The loss function w.r.t. parameters θ_Q in MP-DQN is:

$$L_Q(\theta_Q) = \mathbb{E}\left[\frac{1}{2}\left(y - Q_{kk}(s, k, \mathbf{xe}_k; \theta_Q)\right)^2\right], \tag{1}$$

where $y = r + \gamma \max_{k' \in \mathcal{A}_d} Q_{k'k'}(s', k', \mathbf{xe}_{k'}(s'; \theta_X^-); \theta_Q^-))$ w.r.t. parameters θ_X^-, θ_Q^- for the target networks.

At last, parameters θ_X are updated so as to maximize the sum of Q-values with θ_Q fixed, i.e., the following loss function:

$$L_x(\theta_X) = \mathbb{E}\left[-\sum_{k=1}^{K} Q_{kk}\left(s, k, \mathbf{xe}_k; \theta_Q\right)\right]. \tag{2}$$

3.3 Distributed Training Architectures in DRL

The distributed training paradigm with experience replay has been applied in several popular distributed DRL algorithms, like Gorila [30], Ape-X [18], D4PG [5], and R2D2 [20]. The paradigm contains following components:

- *actor nodes*: run in parallel to generate experiences for the replay buffer,
- *learner node*: learns from the experience replay buffer and periodically updates parameters for corresponding actor nodes,
- *shared replay buffer*: collects experiences from actor nodes and provides training data to the learner node.

The use of a shared experience replay has the advantages of tolerating low latency communications and increasing the sample efficiency.

3.4 Curriculum Learning in DRL

Curriculum learning in DRL is a training methodology that seeks to increase performance or speed up learning of a target task, by considering how best to organize and train on experiences acquired from a series of tasks with different degrees of difficulty. Based on the following assumptions:

- a *task* $t_i = \langle \mathcal{S}_i, \mathcal{A}_i, P_i, R_i \rangle$ is a Markov Decision Process, and $\mathcal{T}$ is a set of tasks.
- $\mathcal{D}^{\mathcal{T}}$ is the set of all possible transition samples from tasks in $\mathcal{T}$:

$$\mathcal{D}^{\mathcal{T}} = \{(s, a, r, s') \mid \exists t_i \in \mathcal{T} \text{ s.t. } s \in \mathcal{S}_i, \\ a \in \mathcal{A}_i, s' \sim P_i(\cdot|s, a), r \leftarrow R_i(s, a, s')\}.$$

Then a *curriculum* can be defined as a *directed acyclic graph* [31]:

$$C = \langle \mathcal{V}, \varepsilon, g, \mathcal{T} \rangle,$$

where $\mathcal{V}$ is the set of vertices, $\varepsilon \subset \{(x, y) \mid (x, y) \in \mathcal{V} \times \mathcal{V} \wedge x \neq y\}$ is the set of directed edges, and $g : \mathcal{V} \to \mathcal{P}(\mathcal{D}^{\mathcal{T}})$ is a function that associates vertices to subsets of samples in $\mathcal{D}^{\mathcal{T}}$, where $\mathcal{P}(\mathcal{D}^{\mathcal{T}})$ is the power set of $\mathcal{D}^{\mathcal{T}}$.

We consider approaches that keep the state and action spaces the same, as well as the environment dynamics, but allow the reward function and initial/terminal state distributions to vary. Inspired by [2, 11, 32], we create a series of tasks, where the agent's initial state distribution starts close to the goal state, and is progressively moved farther away in subsequent tasks, inducing a curriculum of tasks (see Figure 2c).

3.5 Goal-oriented Tasks

Given an MDP problem $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, a goal-oriented task [11] is to reach a goal state in $S^g \subseteq \mathcal{S}$ from a starting state in $S^0 \subseteq \mathcal{S}$. A goal-oriented task is *binary* if its reward function is binary, i.e., $R(s_t) = \mathbb{1}\{s_t \in S^g\}$. It is challenging to solve these tasks in reinforcement learning, since their natural reward functions are sparse and optimizing these sparse reward functions directly is less prone to yielding undesired behaviors. Note that, a scoring task without a goalkeeper in HFO is such a task, which is considered in Section 5.3.

An *adversarial* goal-oriented task generates a goal-oriented task by involving a competitor in the environment. In specific, it is a goal-oriented task in a two-player zero-sum stochastic game [39]. Note that, a scoring task with a goalkeeper in HFO is such a task, which is considered in Section 5.4.

As discussion in previous section, "general curriculum" does not perform well for adversarial goal-oriented tasks, while our approach DPCER can improve the performance. Experiments in

Section 5 show that DPCER is more efficient in both binary and adversarial goal-oriented tasks.

4 DISTRIBUTED PARALLEL CURRICULUM EXPERIENCE REPLAY

In this section, we introduce Distributed Parallel Curriculum Experience Replay (DPCER), a distributed DRL paradigm with parallel curriculum experience replay for goal-oriented tasks. Following the paradigm, we implement a distributed DRL algorithm for PAMDP problems.

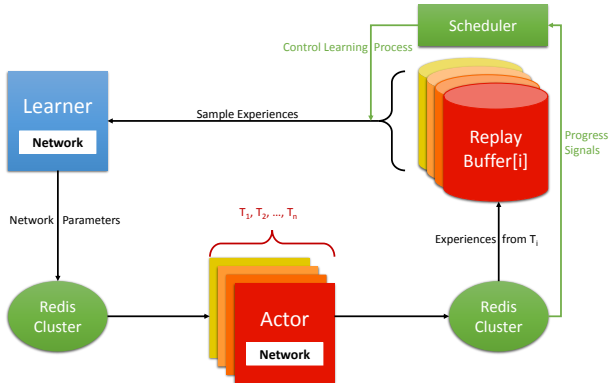


Figure 1: The architecture of DPCER. Actor nodes for the target task T and progressively easier tasks $T_1, \ldots, T_n$ generate experiences to the Redis Cluster. Replay Buffer $[i]$ accesses data coming from task T_i from the Redis. Scheduler accesses progress signals from Actor nodes. Learner node learns from the training data which is selected by Scheduler and updates corresponding network parameters to Redis Cluster.

4.1 Distributed Training Paradigm with Parallel Curriculum Experience Replay

In the distributed setting, we do not follow the original paradigm, like "general curriculum", that trains the agent gradually with a sequence of progressively more difficult tasks step by step. Instead, we use the parallel running actor nodes to generate experiences in this sequence of tasks respectively and train the learner node with experiences chosen by a scheduler node.

The new paradigm depicted in Figure 1 is specified as follows:

- Splitting a goal-oriented task T into a sequence of progressively more difficult tasks $\langle T_1, \ldots, T_n \rangle$. In HFO, T_i specifies the scoring task whose starting position is closer to the goal than the starting position of T_j when $i < j$.
- Creating Actor nodes for tasks $T_1, \ldots, T_n$ and the target task T.
- Using the parallel running Actor nodes to generate experiences for corresponding Replay Buffers and progress signals for Scheduler.
- During each training timestep, choosing experiences in a proper Replay Buffer by Scheduler to train Leaner node.

4.2 Scheduling Polices in DPCER

The scheduling policy for Scheduler is try to maximize Leaner's performance on the original task T by selecting training samples from proper Buffer Replays.

Uniform sampling is a scheduling policy that chooses samples from Buffer Replays uniformly at random, i.e.,

$$p(i) = \frac{1}{N}, \tag{3}$$

where $p(i)$ denotes the probability of choosing sample from Buffer Replay $[i]$ and N denotes the number of Buffer Replays. DPCER with the uniform sampling policy is denoted as DPCER^{us}.

'EXP3'[3] is another scheduling policy which considers the scheduling problem as an N-armed bandit problem [41] and chooses proper Buffer Replays by tracking expected returns of them. In specific, at each time t

$$p(i) = (1-\gamma)\frac{w_t(i)}{\sum_{j=1}^{N} w_t(j)} + \frac{\gamma}{N}, \tag{4a}$$

$$w_{t+1}(i) = w_t(i)\ \exp\left(\frac{\gamma\ signal(i)}{p(i)\ N}\right), \tag{4b}$$

$$signal(i) = r(i)\ step(i), \tag{4c}$$

where γ is the learning rate, $r(i)$ is the profit of choosing Buffer Replay $[i]$ at time t, $step(i) = 1$ if Buffer Replay $[i]$ was chosen at time t and $step(i) = 0$ otherwise. DPCER with the 'EXP3' policy is denoted as DPCER^{exp3}.

4.3 Parametrized Action DRL Algorithms under DPCER

Now we apply DPCER to the popular parameterized action DRL algorithm, MP-DQN, resulting new distributed DRL algorithms in parameterized action space. Without causing confusion, we also name new algorithms as DPCER^{us} and DPCER^{exp3}.

Actor nodes in both DPCER^{us} and DPCER^{exp3} are specified in Algorithm 1, where the actor nodes number n denotes that the target task is split into $n-1$ easier tasks. In particular, n actor nodes are created to act in corresponding environments to generate experiences. Learner node is specified in Algorithm 2. In particular, the Q-network and the action-parameter choosing network are training by experiences chosen by Scheduler from Replay Buffers. Scheduler is specified in Algorithm 3 if 'EXP3' is applied.

5 EXPERIMENTS

In this section, we evaluate the performance of DPCER on the domain of simulated RoboCup soccer. In experiments, we compare DPCER with "general curriculum" on both binary and adversarial goal-oriented tasks. In specific, "Scoring goals without goalkeeper" as illustrated in Figure 2(a) serves as the binary goal-oriented task and "Scoring goals against goalkeeper" as illustrated in Figure 2(b) serves as the adversarial goal-oriented task in experiments. The results show that parallel training of the robot with tasks in the curriculum can improve the performance.

[1]Corresponding source codes are available on line: https://github.com/yuyuguru/Distributed-Parallel-Curriculum-Experience-Replay.

Algorithm 1: Actor nodes

Input: Environment label number n, updating frequency for the network t_{actors}

Initialize ENVIRONMENT$_n$ with label number n ;
$\theta_{Net} \leftarrow$ RedisCluster.GetLearnerNetworks() ;
$t \leftarrow 0$;
while *not Learner is finished* **do**
  $s_0 \leftarrow$ ENVIRONMENT$_n$.Reset() ;
  while *episode is not finished* **do**
    $a_t \leftarrow \theta_{Net}(s_t)$;
    $(r_{t+1}, d_{t+1}, s_{t+1}) \leftarrow$ ENVIRONMENT.Step(a_t) ;
    Transition.Add($\langle s_t, a_t, s_{t+1}, r_{t+1}, d_{t+1}, n\rangle$) ;
    $t = t + 1$;
  RedisCluster.Rpush(Transition) ;
  if $t \mod t_{actors} = 0$ **then**
    RedisCluster.SetSignalProcess($\langle$Signal, $n\rangle$) ;
    $\theta_{Net} \leftarrow$ RedisCluster.GetLearnerNetworks() ;

Algorithm 2: Learner node

Input: Training steps T, all different tasks number N, replay buffer list RBL

$\theta_{Net}, \theta^{-}_{Net} \leftarrow$ InitializeNetwork()
for $i = 1, 2, ..., N$ **do**
  RBL[i] $\leftarrow$ RedisCluster.GetExperience(i) ▷ Run as a threading procedure
for $t = 1, 2, ..., T$ **do**
  Choose task i based on Scheduler's probability. ▷ Equation (3) or (4a) ;
  $\tau \leftarrow$ RBL[i].sample(BatchSize) ;
  $\text{loss}_{Net} \leftarrow \text{Loss}(\tau; \theta_{Net}, \theta^{-}_{Net})$ ▷ Using the loss function in Equation (1) (2) for MPDQN.;
  $\theta_{Net_{t+1}} \leftarrow \text{UpdateParameters}(\text{loss}_{Net}; \theta_{Net_t})$;
  $\theta^{-}_{Net_{t+1}} \leftarrow \text{SoftUpdate}(\theta_{Net_{t+1}})$;
  RedisCluster.Set(θ_{Net});
return θ_{Net}

Algorithm 3: Scheduler with EXP3

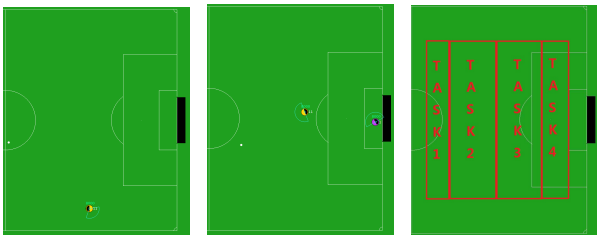
Input: expected weight $w(i) = 1$ for all N tasks

while *Learner node is not finished* **do**
  for $n = 1, 2, ..., N$ **do**
    signals $\leftarrow$ RedisCluster.getSignal(n) ;
    update $w(n)$ with signals ▷ Equation (4b) ;
    compute $p(n)$ ▷ Equation (4a) ;
return p

5.1 Half Field Offense Domain

The RoboCup 2D Soccer Simulation League works with an abstraction of soccer wherein the players, the ball, and the field are all 2-dimensional objects. The state of a HFO example contains 58 continuously-valued features[2], which provides angles and distances to various on-field objects of importance such as the ball, the goal, and the other players. All these features range from −1 to 1. A Full list of state features may be found on HFO's website[3].

The full action space for HFO is: { Dash (power, direction), Turn (direction), Tackle (direction), Kick (power, direction) }, where all the directions are parameterized in the range of $[-180, 180]$ degree and power in $[0, 100]$. There are 4 discrete actions, i.e., Dash, Turn, Tackle, Kick, in this parameterized action space.



(a) Scoring without Goal-keeper (b) Scoring against Goal-keeper (c) Nearby and Far-away Tasks

Figure 2: Goal-oriented Tasks[1].

5.2 Experiment Settings

We apply our algorithms, DPCER^{us} and DPCER^{exp3}, in experiments. We also compare them with algorithms following "general curriculum"[44]. To make it fair, we implemented three DRL algorithms from MP-DQN (similar to DPCER^{us} and DPCER^{exp3}) following "general curriculum" under distributed training paradigm. In specific,

- Distributed Baseline (DB): a baseline DRL algorithm that only implements MP-DQN in the distributed paradigm (APE-X [18]) without curriculum learning.
- Distributed Baseline Curriculum (DBC): a shared experience replay DRL algorithm that implements MP-DQN in the distributed paradigm with "general curriculum" [2] which following the easy-to-hard sequence of tasks.
- Distributed Reverse Curriculum (DRC): a shared experience replay DRL algorithm that refines DBC by incrementally collecting experiences from easier tasks to mitigate the catastrophic forgetting problem, which is inspired by "reverse curriculum" [11].

We also compare our algorithm with a planning algorithm, named Helios, which is programed by Helios [1], the 2012 RoboCup 2D champion team.

5.3 Test 1v0: Scoring Goals without Goalkeeper

In this test, the 2D agent is placed at a random position on the offensive half of the field in the beginning. The task in this test is

[2]Note that, the number of complete features derived from HeliosAgent2D's[1] world model is $58 + 8 \times T + 8 \times O$, where T is the number of teammates and O is the number of opponents.
[3]https://github.com/LARG/HFO/blob/master/doc/manual.pdf

binary goal-oriented. Its reward function is binary, i.e.,

$$r_t = 5\mathbb{I}_t^{goal}. \quad (5)$$

We evaluate the performance of all six algorithms introduced in the experiment setting in Test 1v0. Experimental results are specified in Figure 3, which illustrates the mean episode reward and the mean episode length (the shorter the better) during the training time of these algorithms in 5 cases of the test.

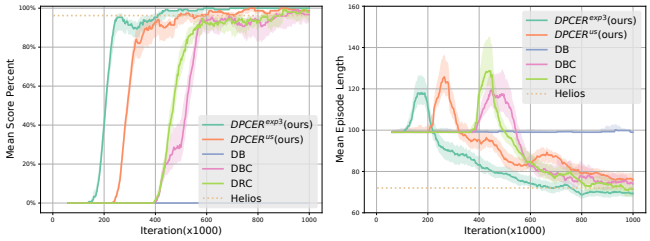


Figure 3: Experimental results for Test 1v0.

The results show that both DPCER and "general curriculum" learned how to score while the Distributed Baseline (DB) failed in the test. Moreover, DPCER is more efficient than "general curriculum", as both DPCER^{us} and DPCER^{exp3} perform better than DBC and DRC. After the training, both DPCER^{us} and DPCER^{exp3} also perform better than Helios.

We also provide the learning curve of Q-value for DBC, DRC, and DPCER^{exp3} in Figure 4. Similar to the performance in Figure 3, the learning curve of Q-value for DPCER^{us} is the same as DPCER^{exp3}. The results show that, compared with "general curriculum", DPCER runs more smoothly and quickly.

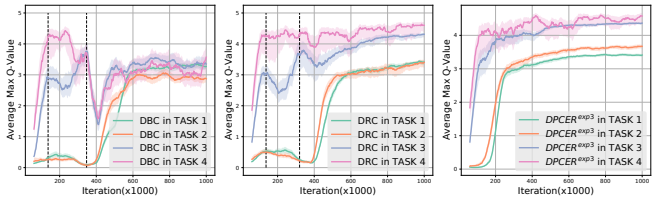


Figure 4: Average of maximum Q-value for Test 1v0.

There are various parameterized action DRL algorithms for this test, while most of them use a reward shaping function as specified in Equation (6). Table 1 summarizes the performance of these algorithms and compares them with ours. The results show that DPCER outperforms others. A demonstration video to illustrate the performance of DPCER^{exp3} in Test 1v0 is available on line[4].

5.4 Test 1v1: Scoring Goals Against Goalkeeper

Scoring goals against a goalkeeper is more challenging, since the goalkeeper's policy is a highly adept code that is programmed by Helios[1], the RoboCup 2D champion team. A long distance shot can easily be blocked by the goalkeeper.

[4]https://youtu.be/ZOhv-KfT8EQ

Table 1: Performance of algorithms on Test 1v0.

Algorithm	Network Iteration	Scoring Percentage	Avg. steps to Goal
Helio[a]	—	96.2%	72
P-DDPG[b]	3M	92.3%	112
a3c.P-DQN[c]	72M	98.9%	81
MP-DQN[d]	2M	91.3%	99
DPCER^{exp3}	1M	**99.5%**	**68**

a. The planning algorithm was programed by Helios [1], the 2012 RoboCup 2D champion team.
b. The performance of P-DDPG algorithm with shaping reward in Equation (6) is from [16] .
c. The performance of a3c.P-DQN algorithm with shaping reward in Equation (6) is from [45].
d. The performance of MP-DQN algorithm with shaping reward in Equation (6) is from [7].
* Our algorithm DPCER^{exp3} with spare reward in Equation (5).

The reward shaping function is specified below, which is adapted by most algorithms [7, 16, 17, 45]:

$$r_t = d_{t-1}(a,b) - d_t(a,b) + \mathbb{I}_t^{kick} + 3\left(d_{t-1}(b,g) - d_t(b,g)\right) + 5\mathbb{I}_t^{goal}. \quad (6)$$

The reward function encourages the agent to approach the ball, i.e., $d(a,b)$ is the distance between the agent and the ball, kick the ball, dribble the ball towards the goal, i.e., $d(b,g)$ is the distance between the ball and the goal, and score a goal.

We also evaluate the performance of all six algorithms in Test 1v1. Experimental results are specified in Figure 5. The results show that both DPCER^{us} and DPCER^{exp3} are efficient and outperform all other DRL algorithms. Notice that, both DPCER and "general curriculum" with incremental curriculum experience replay learned how to score while other DRL algorithms failed in the test. In particular, DB and DBC were stuck at some local optimal solutions, i.e., strategies to approach the ball and dribble towards the goal.

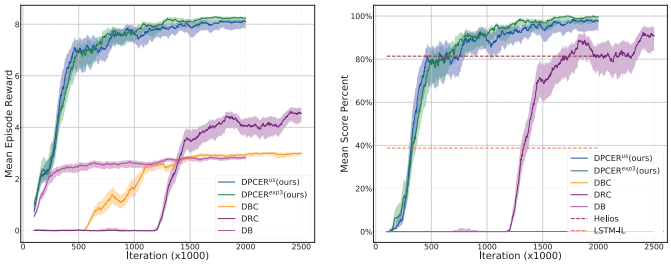


Figure 5: Experimental results for Test 1v1.

We also compare DPCER^{exp3} with other existing algorithms in Table 2. It shows that DPCER^{exp3} is effective and outperforms others, i.e., a planning algorithm and an imitation learning algorithm. A demonstration video that illustrates the performance of DPCER^{exp3} is available on line[5].

5.5 Catastrophic Forgetting Problem in General Curriculum

In this subsection, we discuss the catastrophic forgetting problem for DPCER and "general curriculum" in the training stage.

[5]https://youtu.be/7DnVzkU1WHU

Table 2: Performance of algorithms on Test 1v1.

Algorithm	Network Iteration	Scoring Percentage	Avg. steps to Goal
Helio[a]	–	81.4%	**86**
LSTM[b]	unknown	38.8%	unknown
DPCER^{exp3}	2M	**98.5%**	89

a. The planning algorithm programed by Helios [1].
b. The performance of imitation learning algorithm is from [19].
* Our algorithm DPCER^{exp3} is evaluated by 1000 trials.

In Test 1v0, as illustrated by the black curve on the left sub-figure in Figure 4, the Q-value of DBC for TASK 3 and TASK 4 dropped rapidly. This is due to the catastrophic forgetting problem, as Learner is trained following the easy-to-hard sequence of tasks and the ability on scoring at a nearby position was forgotten during the training process. Although DRC is smoother than DBC, it is still found from the black curve on the middle sub-figure in Figure 4 that DRC is affected by the forgetting problem. DPCER outperforms "general curriculum" in this case.

Similarly, in Test 1v1, as illustrated in Figure 6, DPCER outperforms "general curriculum" as well.

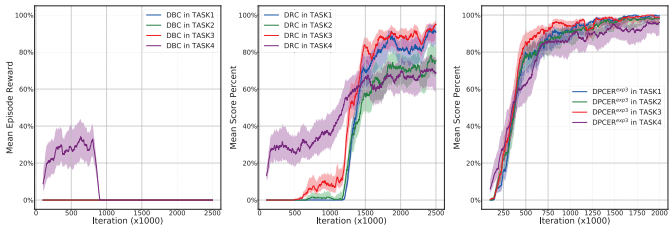


Figure 6: Experimental results for all tasks in Test 1v1.

5.6 Identifying Difficulty of Tasks

In this subsection, we show that DPCER can automatically identify the difficulty of tasks and assign proper sampling probabilities for them in the training stage. As illustrated in Figure 7, DPCER^{exp3} adjusts the sampling probabilities for TASK 1-4 at each iteration in Test 1v0, due to their rewards at the corresponding iteration. In specific, "exp3" has no prior knowledge on four tasks, then their sampling probabilities are assigned equivalently to be 0.25 at the beginning. At the early stage of the training, the rewards for TASK 3 and TASK 4 are increased as both tasks are easier to be learned. Then their sampling probabilities are also increased, while decreasing the probabilities for TASK 1 and TASK 2. After 170,000 iterations, the rewards for TASK 1 and TASK 2 begin to increase. DPCER^{exp3} is aware of the change and properly adjusts the probabilities for tasks. In the end, DPCER^{exp3} considers TASK 1 as the most challenging task and TASK 4 as the simplest task, which matches with our intuition that the task starts farther away from the goal is more difficult, as shown in Figure 2(c).

5.7 Transferring Knowledge among Tasks

As illustrated in Figure 8, we further explore the effect of parallel curriculum experience replay on the early stage of the training in

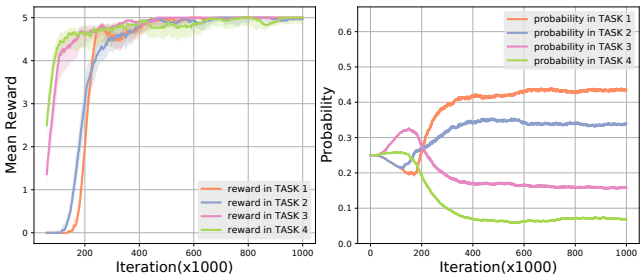


Figure 7: Experimental results of DPCER^{exp3} for all tasks in Test 1v0.

Test 1v1. The learner is trained from experiences in the curriculum replay buffer, which gathers episodes from actors for TASK 1, TASK 2, TASK 3 and TASK 4. The learning curve for TASK 4, i.e., the green curve, in Figure 8 shows that DPCER^{exp3} can learn the knowledge on shooting in TASK 4 quickly. Later, DPCER^{exp3} can learn the knowledge on dribbling towards the goal in TASK 3. A video illustrating the phenomenon is available on YouTube[6]. DPCER allows the learner to transfer the knowledge on shooting and dribbling to Task 1 and Task 4, which enables the agent to dribble towards the goal and make a scoring shot.

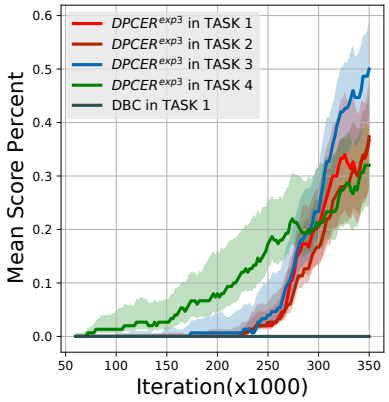


Figure 8: Early stage of the training for all tasks in Test 1v1.

6 CONCLUSION

In this paper, we show that the parallel training of the robot with tasks in the curriculum can improve the performance of DRL algorithms. We introduce Distributed Parallel Curriculum Experience Replay (DPCER), a distributed training paradigm with parallel curriculum experience replay for goal-oriented tasks. Following the paradigm, we propose two distributed DRL algorithms, DPCER^{us} and DPCER^{exp3}, in parameterized action space. We test new algorithms on the domain of simulated RoboCup soccer. Experimental results show that, our algorithms are effective in both binary and adversarial goal-oriented tasks.

[6] https://youtu.be/e78opuZ8Vjc

ACKNOWLEDGMENTS

The work is partially supported by the National Major Program for Technological Innovation 2030 – New Generation Artificial Intelligence (No. 2018AAA0100500), CAAI-Huawei MindSpore Open Fund, and Anhui Provincial Development and Reform Commission 2020 New Energy Vehicle Industry Innovation Development Project "Key System Research and Vehicle Development for Mass Production Oriented Highly Autonomous Driving".

REFERENCES

[1] Hidehisa Akiyama and Tomoharu Nakashima. 2013. Helios base: An open source package for the robocup soccer 2d simulation. In *Robot Soccer World Cup*. 528–535.

[2] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. 1996. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning* 23, 2-3 (1996), 279–303.

[3] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.

[4] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. 2016. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256* (2016).

[5] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. 2018. Distributed distributional deterministic policy gradients. In *ICLR*.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *ICML*. 41–48.

[7] Craig J. Bester, Steven D. James, and George D. Konidaris. 2019. Multi-Pass Q-Networks for Deep Reinforcement Learning with Parameterised Action Spaces. *arXiv preprint arXiv:1905.04388* (2019).

[8] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[10] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *ICML*.

[11] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. 2017. Reverse Curriculum Generation for Reinforcement Learning. In *CoRL*. 482–495.

[12] Haotian Fu, Hongyao Tang, Jianye Hao, Zihan Lei, Yingfeng Chen, and Changjie Fan. 2019. Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces. In *IJCAI*.

[13] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. 2017. Automated curriculum learning for neural networks. In *ICML*. 1311–1320.

[14] Danijar Hafner, James Davidson, and Vincent Vanhoucke. 2017. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878* (2017).

[15] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. 2016. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

[16] Matthew Hausknecht and Peter Stone. 2016. Deep Reinforcement Learning in Parameterized Action Space. In *ICLR*.

[17] Matthew Hausknecht and Peter Stone. 2016. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*.

[18] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. 2018. Distributed prioritized experience replay. In *ICLR*.

[19] Ahmed Hussein, Eyad Elyan, and Chrisina Jayne. 2018. Deep Imitation Learning with Memory for Robocup Soccer Simulation. In *EANN*. 31–43.

[20] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. 2019. Recurrent experience replay in distributed reinforcement learning. In *ICLR*.

[21] Mehdi Khamassi, George Velentzas, Theodore Tsitsimis, and Costas Tzafestas. 2017. Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task. In *IRC*. 28–35.

[22] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. 1997. The RoboCup Synthetic Agent Challenge 97. In *IJCAI*. 24–29.

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. In *NIPS*.

[24] Patrick MacAlpine and Peter Stone. 2018. Overlapping layered learning. *Artificial Intelligence* 254 (2018), 21–43.

[25] Warwick Masson, Pravesh Ranchod, and George Konidaris. 2016. Reinforcement learning with parameterized actions. In *AAAI*. 1934–1940.

[26] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2019. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems* (2019).

[27] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[28] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*. 1928–1937.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[30] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. 2015. Massively parallel methods for deep reinforcement learning. In *ICML*.

[31] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *arXiv preprint arXiv:2003.04960* (2020).

[32] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. 2016. Source task creation for curriculum learning. In *AAMAS*. 566–574.

[33] OpenAI. 2018. OpenAI Five. https://blog.openai.com/openai-five/.

[34] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. 2016. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 81:1–81:12.

[35] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. 2017. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073* (2017).

[36] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. 2019. Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments. *CoRL* (2019).

[37] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the Atari Human Benchmark. *arXiv* (2020), arXiv–2003.

[38] Alessandra Sciutti, Ambra Bisio, Francesco Nori, Giorgio Metta, Luciano Fadiga, and Giulio Sandini. 2013. Robots can be perceived as goal-oriented agents. *Interaction Studies* 14, 3 (2013), 329–350.

[39] Yoav Shoham and Kevin Leyton-Brown. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

[40] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[41] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement Learning: An Introduction*. MIT press.

[42] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. 2019. AlphaStar: Mastering the real-time strategy game StarCraft II. *DeepMind Blog* (2019).

[43] Oriol Vinyals, Igor Babuschkin, Wojciech Marian Czarnecki, Michael Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard E Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[44] Yuxin Wu and Yuandong Tian. 2017. Training Agent for First-Person Shooter Game with Actor-Critic Curriculum Learning. In *ICLR*.

[45] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. 2018. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394* (2018).