FedPBT: Leveraging Federated Learning for Efficient Hyperparameter Schedule Search

Anonymous Author(s)
Affiliation
Address
email

Abstract

Population Based Training (PBT) searches over a large space of hyperparameter schedules in a single time-efficient experiment run. However, the amount of computation it requires scales linearly with the population size N. We observe that in reinforcement learning settings, the sample complexity of PBT can potentially be reduced N-fold by sharing experiences between population members. We realize this efficiency gain with FedPBT, a variant of PBT that draws on ideas from federated learning to enable experience sharing for on-policy algorithms. Our evaluation shows that FedPBT achieves near N-fold computational savings without any efficiency loss in many reinforcement learning training scenarios, yielding faster training and better outcomes than any of PBT, federated learning, or independent trial execution alone.

1 Introduction

Deep reinforcement learning has made significant progress in a variety of applications including robotics and graphics, but their training is highly sensitive to the choice of hyperparameters [6]. With many state-of-the-art algorithms depending on a significant number of hyperparameters [17], it comes as no surprise that deep learning practitioners are willing to go to great lengths to find the optimal hyperparameters. For deep reinforcement learning specifically, wherein the agent navigates a dynamic problem, the optimal hyperparameters themselves are dynamic and the problem of finding an optimal hyperparameter *schedule* becomes even more computationally intensive and time-consuming. Fortunately, approaches such as Population Based Training [7] offer the promise of simultaneously training and discovering good hyperparameter schedules.

Population Based Training is a training procedure that utilizes N copies of the training problem, each with different hyperparameters, varied across training. Though PBT has been shown to achieve state-of-the-art results, it necessarily increases sample complexity N-fold, where N is the population size [8]. Since deep RL is typically bottle-necked by the sampling process (e.g., interaction with high-fidelity software simulators [4], physical robots, other real-world systems) and not by the learning process [18] (which is typically fast enough given the small size of neural networks), this motivates us to improve PBT by creating an approach which is sample-efficient.

Given a fixed pool of resources for training, today a user has to make a choice: run a single large training run with a single hyperparameter (i.e., using a training procedure such as distributed synchronous SGD), run smaller independent trials with different hyperparameters (i.e., grid search), or adopt some population-based algorithm (i.e., PBT). Unless the optimal hyperparameters are known ahead of time, it is almost certain that a sweep over multiple hyperparameters or application of PBT will yield better results than dedicating all resources to a single training run for longer duration. However, there is the possibility of doing better with a hybrid approach between large-scale synchronous training and PBT.

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

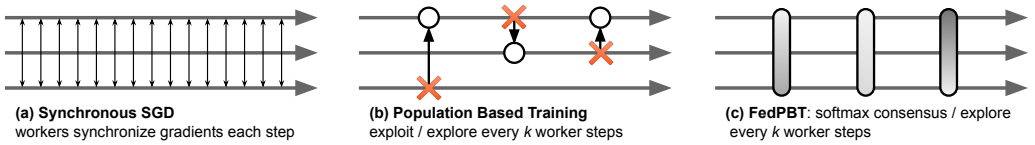

Figure 1: FedPBT leverages softmax policy consensus to keep population weights in sync while allowing for exploration of the hyperparameter space between population members. This is in contrast to unmodified PBT, which hard clones weights from high-performing to lower-performing trial, and also Synchronous SGD, which averages gradients from each worker with uniform weighting.

In this paper we introduce FedPBT, such a hybrid algorithm that leverages gradient-sharing techniques from federated learning [14] to improve the sample efficiency of PBT. While FedPBT is applicable to supervised settings, for reinforcement learning in particular it enables order-of-magnitude sample efficiency improvements through *sample sharing*. FedPBT is simple and works for any RL algorithm including on-policy algorithms. In our evaluation, we show that FedPBT greatly outperforms grid search, standard PBT, and naive federated learning baselines across a variety of RL tasks.

2 Background and Overview

Here we describe the relevant background of Deep Reinforcement Learning (Deep RL), Population Based Training (PBT) and Federated Learning. We then overview our method, Federated Population Based Training (FedPBT) which is specific to Deep Reinforcement Learning.

Deep Reinforcement Learning: One categorization of deep reinforcement learning is by their training and sampling interaction: either on-policy or off-policy. On-policy algorithms such as Vanilla Policy Gradients [21] and PPO [17] assumes that the samples are taken from the current policy being optimized. Off-policy algorithms such as DQN [15], on the other hand, do not make this assumption. On-policy algorithms are often preferred for their reliability and speed, while off-policy algorithms can trivially utilize experience replay [15] and sample sharing. Employing experience replay (and therefore sample sharing) in on-policy approaches is an ongoing area of research [5].

Population Based Training (PBT): Population Based Training (Figure 1(b)) is an asynchronous optimization algorithm that jointly optimizes a population of models and their hyperparameters. Importantly, PBT discovers a *schedule* of hyperparameter settings rather than following the generally sub-optimal strategy of trying to find a single fixed set to use for the whole course of training. In PBT, N population members are trained for some interval. The population is sorted by a user-defined *fitness* measure, and members of the lowest-performing quantile copy weights from the best-performing quantile and then mutate hyperparameters. In the context of Deep RL, fitness can be determined by any criterion, commonly the mean reward collected over a smoothed set of episodes. As aforementioned, PBT induces N-fold sample complexity where N is the population size.

To make PBT sample efficient, we can try to reuse samples between members of the population. This is feasible for off-policy algorithms, but not so for on-policy (i.e. PPO). As we show, naively sample sharing in PBT-PPO does not produce good results since the policies diverge so much that they cannot learn well from each other's samples. Although optimal weights propagate quickly in PBT, by design they do not propagate quickly enough to eliminate potential policy divergence.

Federated Learning: Federated learning is a cooperative stochastic gradient descent algorithm which enables multiple parties to collaboratively learn a model over their disjoint datasets [14]. It has been deployed at scale by Google, and enjoyed success as a communication-efficient improvement to synchronous SGD [3]. In each generation, N population members are trained for some perturbation interval. Each population member has a fitness measure, which is almost always the dataset magnitude. Rather than preserving any individual well-performing population members, a fitness-weighted average of the models is performed and the resulting model redistributed to each population member. Similar policy consensus steps have been employed in multi-agent reinforcement learning [24, 23] but not for population based optimization.

The convergence of federated learning on supervised learning tasks is well studied [22] as a percentage of the best attainable performance by a centralized model. Without using further exploration/evolution

Table 1: Comparison of FedPBT with standard Population Based Training, Synchronous SGD (including federated variants), and independent trial execution. Since RL training is often bottlenecked by sampling complexity in practice, FedPBT achieves up to a N-fold speedup over standard PBT search, which in turn more efficiently explores hyperparameters than running trials independently.

Training Strategy	**SGD Time**	**Sample Complexity**	**Search Space**	**Compatibility**
N Independent Trials	$O(N \times T)$	$O(N \times T)$	fixed hyperparams	any
Synchronous SGD, Federated SGD	$O(N \times T)$	$O(N \times T)$	none	any
Population Based Training	$O(N \times T)$	$O(N \times T)$	**schedules**	any
FedPBT (ours)	$O(N \times T)$	$\mathbf{O(T)}$	**schedules**	RL training

strategies [20, 25] federated learning can never do better than a single learner in supervised learning. However, the online nature of reinforcement learning means it is theoretically possible for a federated learning strategy to outperform the single actor. Other work in federated RL [26, 13] use the Q-function or value function as a measure of fitness and are not altogether related to our approach.

Federated Population Based Training: In distributed synchronous SGD (i.e., Figure 1(a)), multiple workers compute gradients over data in the same way, and synchronize gradients with each other at each step of the optimization process. This ensures all models are kept in sync. However, this isn't the only way to synchronize the models. In federated learning [14], the gradients of each worker are weighted by dataset magnitude and averaged to ensure synchronization.

Our first observation is that there is another way to weight the importance of gradients—by considering the objective improvement they provide to the model. For example, a certain series of updates may improve the validation accuracy of a model, or the mean reward achievable by a trained RL policy. We propose weighting and combining worker gradients via *softmax-based gradient synchronization*, which takes a softmax over the objective value improvement with some temperature β.

Softmax-based gradient synchronization keeps the parameters of each worker closely in sync, only allowing drift between the synchronization intervals and ensuring that sample-sharing is not useless. This is in contrast with the standard PBT algorithm, in which the weights of population members may drift arbitrarily far from each other so long as they remain competitive in performance. While this diversity is a desirable property, it is not *necessary* for hyperparameter exploration. For example, it is possible to explore the impact of different learning rates (i.e., discover an LR schedule), without significant parameter diversity between models.

Our key observation: A population of N reinforcement learning agents can share experiences between each other if their weights are kept in sync, even if they are following different training hyperparameters. This provides three simultaneous benefits. First, by sharing samples we reduce the sample complexity of training by N-fold (Table 1). Since N typically varies from 8-16, this is a considerable improvement. Second, by using periodic soft gradient-based synchronization, we keep SGD efficiency loss to a minimum. Finally, with each agent following different hyperparameters and the use of a PBT-like `explore()` mechanism, we can efficiently discover hyperparameter schedules during the course of training.

3 Implementation

In this section, we describe FedPBT, our variant of PBT that utilizes *gradient sharing* and *sample sharing* for significant efficiency gains. In contrast to distributed off-policy RL approaches that already naturally share experiences between multiple actor-learner pairs, *sample sharing* is difficult to utilize in on-policy learning since parallel actor-learners will quickly diverge so far that sharing experiences only harms the learning process. *Gradient sharing*, as used in federated learning for synchronization or even synchronous SGD, ensures that a population of policies does not diverge greatly.

Sample Sharing: By sharing samples, we can eliminate the N-fold sample complexity introduced by vanilla PBT. Rather than each population member computing an entire training batch of samples,

each population member computes $1/N$ this many samples and the samples are replicated across the GPUs which will be used to train the population. Then, each population member takes some number of optimizer steps on the collectively gathered samples. This is one "round" of training, which can contain multiple epochs, and multiple rounds are performed in each generation.

Gradient Sharing: By synchronizing weights regularly, we can ensure that members of the population do not diverge so much that they cannot learn from each other's samples. However, when synchronization is done with a uniform average good agent are not rewarded and reaching good optima takes longer (or doesn't happen at all), and when synchronization is done with greedy max, policies often diverge entirely.

At the end of a generation of the PBT procedure, reward is used as the fitness measure to perform a softmax fitness-weighted average of gradients. The resulting gradient captures the updates performed by the best-performing policies. Since the policies trained on the same samples, the aggregated gradient can be applied to the last synchronized model and then redistributed to every member of the population. We call this *softmax policy consensus*, and it acts as a middle ground between uniform averaging and greedy max which, at the appropriate temperature β, propagates good policies while leaving room to explore, and crucially enables sample sharing.

When $\beta \to 0$, we obtain a synchronous optimization process similar to distributed synchronous SGD. On the flip side, as $\beta \to \infty$, the procedure becomes similar to Population Based Training [7], where the weights of top-performing members (i.e., workers) of the population are copied to other workers. Softmax averaging puts more weight on exploitation than the uniform averaging update used in synchronous SGD. As the results show, this uniform averaging does not achieve the same convergence as single actor. Softmax averaging also puts more weight on exploration than greedy max, which performs poorly because it does not give less fit members of the population enough time to grow, and often dives quickly into suboptimal local minima.

Putting things together: By combining PBT (Algorithm 1) and Federated Training (Algorithm 2), we arrive at FedPBT (Algorithm 3). We use the same hyperparameter exploration step as PBT, wherein the worst-performing hyperparameters of the population are replaced by the best-performing hyperparameters. As with PBT, perturbation interval choice is important, and we use the same perturbation intervals for PBT and FedPBT.

Algorithm 1 Population Based Training

procedure TRAIN(P) ▷ initial population P
  for $(\theta, h, p, t) \in P$ (asynchronous in parallel) **do**
    while not end of training **do**
      $\theta \leftarrow \texttt{step}(\theta|h)$ ▷ one step of optimization using hyperparameters h
      $p \leftarrow \texttt{eval}(\theta)$ ▷ current model evaluation
      if ready(p, t, P) **then**
        $h^{'}, \theta^{'} \leftarrow \texttt{exploit}(h, \theta, p, P)$ ▷ use the rest of population to find better solution
        if $h \neq h^{'}$ **then**
          $h, \theta \leftarrow \texttt{explore}(h^{'}, \theta^{'}, P)$ ▷ produce new hyperparameters h
          $p \leftarrow \texttt{eval}(\theta)$ ▷ new model evaluation
        update P with new $(\theta, h, p, t+1)$ ▷ update population
  return θ with the highest p in P

4 Evaluation

We seek to answer the following questions in our evaluation:

1. How does FedPBT compare to grid search (both sequential and parallel) and normal PBT in terms of sample complexity?
2. Does on-policy FedPBT perform well with only sample sharing, or is gradient sharing necessary as well?
3. How sensitive is FedPBT to the softmax temperature hyperparameter (i.e., did we just shift the search problem to this new hyperparameter), and to population size?

Algorithm 2 Federated Training

procedure TRAIN(P) ▷ initial population P
  for $(\theta, h, p, t) \in P$ (synchronous in parallel) **do**
    while not end of training **do**
      $\theta \leftarrow \texttt{step}(\theta|h)$ ▷ one step of optimization using hyperparameters h
      $p \leftarrow \texttt{eval}(\theta)$ ▷ dataset magnitude the model was trained on
      if allready(p, t, P) **then**
        $\theta^{'} \leftarrow \texttt{update}(\beta, \theta, p, P)$ ▷ come to consensus with population on better weights
      update P with new $(\theta, h, p, t+1)$ ▷ update population
  return θ ▷ global model
procedure UPDATE(θ, p, P) ▷ weights and dataset magnitudes of entire population
  denom $\leftarrow \sum_P p$
  for all $(\theta, p) \in P$ **do**
    $\theta^{'} \leftarrow \frac{p}{denom} \times \theta$ ▷ dataset-magnitude weighting
  return $\sum_P \theta^{'}$ ▷ consensus

Algorithm 3 Federated Population Based Training

procedure TRAIN(P, β) ▷ initial population P, temperature β
  for $(\theta, h, p, t) \in P$ (synchronous in parallel) **do**
    while not end of training **do**
      $\theta \leftarrow \texttt{step}(\theta|h)$ ▷ one step of optimization using hyperparameters h
      $p \leftarrow \texttt{eval}(\theta)$ ▷ current policy reward
      if allready(p, t, P) **then**
        $\theta^{'} \leftarrow \texttt{update}(\beta, \theta, p, P)$ ▷ come to consensus with population on better weights
        $h^{'} \leftarrow \texttt{exploit}(h, \theta, p, P)$ ▷ use population to find better hyperparams
        if $h \neq h^{'}$ **then**
          $h, \theta \leftarrow \texttt{explore}(h^{'}, \theta^{'}, P)$ ▷ produce new hyperparameters h
          $p \leftarrow \texttt{eval}(\theta)$ ▷ new model evaluation
        update P with new $(\theta, h, p, t+1)$ ▷ update population
  return θ ▷ global model
procedure UPDATE(β, θ, p, P) ▷ weights and evaluation results of entire population
  denom $\leftarrow \sum_P \exp\beta \times p$
  for all $(\theta, p) \in P$ **do**
    $\theta^{'} \leftarrow \frac{\exp\beta \times p}{denom} \times \theta$ ▷ softmax-weighting
  return $\sum_P \theta^{'}$ ▷ consensus

4.1 Experimental Setup

We evaluate FedPBT using Proximal Policy Optimization as the underlying RL algorithm. This evaluates the challenging case of sharing samples between a population of on-policy algorithms. To evaluate robustness across a range of environments, we test against the following five gym environments: `MountainCarContinuous-v0`, `HalfCheetah-v2`, `Humanoid-v2`, `Hopper-v2`, and `BreakoutNoFrameskip-v4`. We implemented FedPBT on top of Tune [12], and use the PPO implementation and initial hyperparameters from RLlib [11] for parallel training. Each trial configuration is evaluated four times, and we report the best of each. We run each trial until the reward plateaus, and report summary results in tables below (the full training curves are available in the Appendix).

Initial Hyperparameters: Because we are evaluating hyperparameter optimization algorithms, we do not evaluate only the best known hyperparameter configuration. Instead, we initialize each algorithm with the same spread of hyperparameters combinations, which includes the known optimal configuration. The hyperparameter tuned is the *learning rate*, and we allow selection of values in the set of `{1e-2, 5e-3, 1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 5e-6}`. The exact hyperparameter configurations and learning rate ranges used for each scenario can be found in the Appendix. We note that some of these configurations are not tuned for timestep efficiency, rather, they are tuned for

Table 2: Comparison of FedPBT with parallel grid search, sequential grid search, the best trial chosen from the grid search, standard Population Based Training on five different environments. FedPBT achieves the best performance of any of the search techniques, nearly matching or exceeding the cherry-picked best grid search trial.

Environment	**Timesteps to:**		**Reward at timestep:**		
`Breakout`	70 reward	140 reward	1M	10M	Max
Parallel grid search	16M	56M	6 rew	28 rew	200 rew
Sequential grid search	11.5M	56M	10 rew	80 rew	200 rew
Best of grid search	4M	14M	20 rew	110 rew	200 rew
PBT	11M	27M	6 rew	20 rew	200 rew
FedPBT ($\beta = 4$)	**3M**	**6M**	**30 rew**	**140 rew**	200 rew
`MountainCar`	90 reward		80K		Max
Parallel grid search	448K		-6 rew		90 rew
Sequential grid search	100K		-20 rew		90 rew
Best of grid search	56K		90 rew		90 rew
PBT	352K		-49 rew		90 rew
FedPBT ($\beta = 1.5$)	**50K**		**90 rew**		90 rew
`HalfCheetah`	4000 reward	8000 reward	10M	100M	Max
Parallel grid search	40M	240M	750 rew	6000 rew	8000 rew
Sequential grid search	77.5M	100M	2500 rew	5500 rew	8000 rew
Best of grid search	**10M**	60M	**4000 rew**	8000 rew	8000 rew
PBT	72M	260M	800 rew	5200 rew	9000 rew
FedPBT ($\beta = 1.5$)	17M	**54M**	3500 rew	**9200 rew**	**9200 rew**
`Humanoid`	3000 reward	5000 reward	10M	100M	Max
Parallel grid search	224M	316M	273 rew	630 rew	5800 rew
Sequential grid search	76M	95M	400 rew	3500 rew	5800 rew
Best of grid search	**52M**	**70M**	440 rew	**5500 rew**	5800 rew
PBT	210M	301M	100 rew	398 rew	5600 rew
FedPBT ($\beta = 3$)	63M	109M	**460 rew**	4800 rew	**6100** rew
`Hopper`	1500 reward	3000 reward	2M	20M	Max
Parallel grid search	22.8M	60M	60 rew	2400 rew	3200 rew
Sequential grid search	10.8M	50M	162.5 rew	1950 rew	3200 rew
Best of grid search	5.7M	12.5M	300 rew	3000 rew	3200 rew
PBT	21.2M	48M	170 rew	2220 rew	3200 rew
FedPBT ($\beta = 3$)	**4.5M**	**10.5M**	**380 rew**	**3100 rew**	3200 rew

wall-time performance. However, we nevertheless believe they are a reasonable initializations for our evaluation since we are evaluating the search strategy and not the RL algorithm itself. Both PBT and FedPBT use a perturbation interval of 4-8 PPO training iterations depending on the task (targeting $\sim$120 seconds per perturbation), a population size of 4, and a resampling probability of 0.25.

Non-Triviality: To check the non-triviality of each experiment, we include the grid search baseline. This ensures that we are testing scenarios where the extra compute cost of PBT is being effectively leveraged for hyperparameter exploration, and is not just redundant effort. It is expected that baseline PBT outperforms parallel grid search, because it is able to more efficiently explore the hyperparameter space. Indeed, our results show that PBT can sometimes beat even the *best* of the grid search trials (e.g., in `Humanoid` and `HalfCheetah`), showing the non-triviality of the setup and importance of schedule learning.

4.2 Comparison to Baselines

In Table 2 we compare FedPBT against parallel grid search, the lone best trial from the grid search, parallel grid search, the expected reward given sequential grid search, and standard PBT. Compared to other search techniques, FedPBT achieves the lowest number of timesteps to given rewards, and also the highest final rewards in all cases. It also beats the artificial "best grid search trial" results

Table 3: Ablation of gradient sharing and softmax temp on BreakoutNoFrameskip-v4

Training Strategy	Timesteps to reach 70 reward	Timesteps to reach 140 reward	Reward at 1M timesteps	Reward at 10M timesteps
Parallel grid search	20M	56M	6 rew	28 rew
Best of grid search	5M	14M	20 rew	110 rew
PBT	11M	27M	6 rew	20 rew
PBT-SS	>100M	>100M	15 rew	45 rew
FedPBT ($\beta = 4$)	**3M**	**6M**	**30 rew**	**140 rew**
FedPBT ($\beta = 0$)	>100M	>100M	18 rew	55 rew

Table 4: Ablation of sample-sharing (PBT-GS) and different softmax temperatures on HalfCheetah-v2.

Training Strategy	Timesteps to reach 4000 reward	Timesteps to reach 8000 reward	Reward at 10M timesteps	Reward at 100M timesteps
Parallel grid search	40M	240M	750 rew	6000 rew
Best of grid search	**10M**	60M	**4000 rew**	6000 rew
PBT	72M	260M	800 rew	5200 rew
PBT-GS	80M	400M	600 rew	4800 rew
FedPBT (N=4, $\beta = 0.0$)	21M	>400M	2500 rew	6500 rew
FedPBT (N=4, $\beta = 0.5$)	21M	>400M	2700 rew	6500 rew
FedPBT (N=4, $\beta = 1.0$)	19M	88M	2900 rew	8300 rew
FedPBT (N=4, $\beta = 1.5$)	11M	**54M**	3500 rew	**9200 rew**
FedPBT (N=4, $\beta = 2.0$)	19M	100M	2700 rew	8000 rew

for `MountainCar`, `Hopper`, `Breakout`, and `HalfCheetah`, which is possible since PBT in general learns schedules of hyperparameters that may be more effective than any single fixed value. We find that *FedPBT almost eliminates the sample overhead of PBT across a wide variety of tasks*.

4.3 Impact of Proposed Techniques

In Table 3, we study the necessity of gradient sharing and softmax policy consensus. First, we consider PBT with just sample-sharing added (PBT-SS). While this implementation is able to reach reasonable rewards towards the beginning of training, it never reaches the even 70 reward after 100M timesteps due to the divergence of the policies. However, adding gradient sharing with FedPBT stabilizes training and outperforms the other baselines. We also consider FedPBT with uniform sample weighting (softmax temperature $\beta = 0$). This effectively disables the evolutionary aspect of PBT and reduces it to synchronous SGD, and as the table shows, this also is not able to reach higher rewards on this task within 100M steps.

We find these results are general—FedPBT requires both sample sharing and gradient sharing to stabilize training and sufficiently explore the hyperparameter space. We note that it is likely off-policy algorithms (or using importance sampling correction techniques) could alleviate the need for gradient sharing. Our evaluation only considers the on-policy setting without importance correction, a strictly harder scenario.

4.4 Sensitivity to Hyperparameters

Finally, in Tables 4 and 5 we analyze the impact of the FedPBT softmax temperature and PBT population size. We find that FedPBT does reasonably well across a broad range of softmax temperatures, consistently achieving higher rewards than PBT at 100M timesteps. However, FedPBT sometimes does not converge to the highest rewards if the softmax temperature is set too low, since this effectively degrades FedPBT to synchronous SGD, disabling exploration. We also show that FedPBT is robust across a range of population sizes, providing speedups for $N = 4$ to $N = 20$.

Table 5: Ablation of different population sizes on MountainCarContinuous-v0.

Training Strategy	Timesteps to reach 90 reward	Reward at 80K timesteps
Parallel grid search (N=8)	448K	-6 rew
Best of grid search	56K	**90 rew**
PBT (N=4)	352K	-49 rew
PBT (N=8)	732K	-36 rew
PBT (N=20)	1260K	-50 rew
FedPBT (N=4)	93K	87.5 rew
FedPBT (N=8)	112K	82.5 rew
FedPBT (N=20)	**50K**	**90 rew**

4.5 Summary

FedPBT performs as well as or better than PBT on a range of problems, but critically the sample complexity of FedPBT does not scale linearly with the population size. When no hyperparameter exploration is necessary, such as when the optimal hyperparameters are known, FedPBT performs similarly to PPO while being as parallelizable as synchronous SGD. Full graphs with error bars can be found in the appendix; we note that all three approaches have high variance. The ablation studies validate the importance of the techniques used in FedPBT. FedPBT without gradient sharing (PBT-SS) performs poorly because the policies diverge too much to make sample sharing helpful. FedPBT without sample sharing (PBT-GS) is not sample-efficient because experiences are not replicated.

FedPBT does have downsides which PBT does not. The reduced population diversity in parameter-space (as opposed to hyperparameter-space), while key to achieving efficient hyperparameter schedule search with sample sharing, can potentially inhibit generalization to more complex tasks, since it collapses the population to one mode of behaviour. Nevertheless, we believe FedPBT to be useful for a common case of Population Based Training—to discover schedules for simple agent hyperparameters.

5 Related Work

Hyperparameter Optimization: Bayesian methods and early stopping methods are commonly used approaches for hyperparameter optimization. Bayesian optimization often utilizes each training run with a particular set of hyperparameters to decide subsequent hyperparameters searched [19, 2, 1]. For these methods to perform well, multiple model training evaluations need to take place sequentially, which can be time-consuming. Early stopping mechanisms [9, 10] have limited use when optimizing over hyperparameter schedules, which may have performance improvements only shown in later stages of training. FedPBT and other population-based training algorithm can leverage partially trained models to accelerate learning and to optimize over learning rate schedules.

Hyperparameter Optimization for RL: To the best of our knowledge, there is no comparable algorithm to FedPBT that simultaneously offers sample efficiency at the level of a single trial and the ability to also explore the hyperparameter space. The closest related work is HOOF [16], which attempts to select hyperparameters yielding the best estimated 1-step improvement on each optimization step. In contrast, FedPBT optimizes directly against the empirical objective, yielding a much simpler procedure that requires no extra computations.

6 Conclusion

We present FedPBT, an efficient variant of PBT that leverages ideas from federated learning to almost eliminate the sample complexity overhead incurred by maintaining a population of agents. We believe that FedPBT is a superior alternative to PBT in many practical scenarios where efficient hyperparameter exploration is important. Our evaluation shows that FedPBT outperforms PBT in challenging hyperparameter schedule search scenarios across several benchmarks.

Bibliography

[1] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

[2] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

[3] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, H. B., Overveldt, T. V., Petrou, D., Ramage, D., and Roselander, J. (2019). Towards federated learning at scale: System design.

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

[5] Han, S. and Sung, Y. (2017). Amber: Adaptive multi-batch experience replay for continuous action control.

[6] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017). Deep reinforcement learning that matters.

[7] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population based training of neural networks.

[8] Li, A., Spyra, O., Perel, S., Dalibard, V., Jaderberg, M., Gu, C., Budden, D., Harley, T., and Gupta, P. (2019). A generalized framework for population based training.

[9] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization.

[10] Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. (2018). Massively parallel hyperparameter tuning.

[11] Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. (2017). Rllib: Abstractions for distributed reinforcement learning.

[12] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. (2018). Tune: A research platform for distributed model selection and training.

[13] Liu, B., Wang, L., and Liu, M. (2019). Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems.

[14] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2016). Communication-efficient learning of deep networks from decentralized data.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

[16] Paul, S., Kurin, V., and Whiteson, S. (2019). Fast efficient hyperparameter tuning for policy gradients. *arXiv preprint arXiv:1902.06583*.

[17] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[18] Serban, I. V., Sankar, C., Pieper, M., Pineau, J., and Bengio, Y. (2018). The bottleneck simulator: A model-based deep reinforcement learning approach.

[19] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

[20] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning.

[21] Thomas, P. S. and Brunskill, E. (2017). Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines.

[22] Wang, J. and Joshi, G. (2018). Cooperative sgd: A unified framework for the design and analysis of communication-efficient sgd algorithms.

[23] Zhang, K., Yang, Z., Liu, H., Zhang, T., and Başar, T. (2018). Fully decentralized multi-agent reinforcement learning with networked agents.

[24] Zhang, Y. and Zavlanos, M. M. (2019). Distributed off-policy actor-critic reinforcement learning with policy consensus.

[25] Zhu, H. and Jin, Y. (2018). Multi-objective evolutionary federated learning.

[26] Zhuo, H. H., Feng, W., Xu, Q., Yang, Q., and Lin, Y. (2019). Federated reinforcement learning.