SoftPBT: Leveraging Experience Replay 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 SoftPBT, a variant of PBT that uses *softmax policy consensus* to enable experience sharing for on-policy algorithms. Our evaluation shows that SoftPBT achieves near N-fold computational savings without any efficiency loss in many reinforcement learning training scenarios, yielding faster training and better outcomes than state-of-the-art hyperparameter optimization algorithms such as PBT and HyperOpt.

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 [5]. With many state-of-the-art algorithms depending on a significant number of hyperparameters [15], 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 [6] 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 [7]. Since deep RL is typically bottle-necked by the sampling process (e.g., interaction with high-fidelity software simulators [3], physical robots, other real-world systems) and not by the learning process [16] (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 set of hyperparameters (i.e., using a training procedure such as distributed synchronous SGD) while sharing experiences between workers, run smaller independent trials with different hyperparameters (i.e., grid search) without experience sharing, or adopt some population-based algorithm (i.e., PBT) without experience sharing. 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

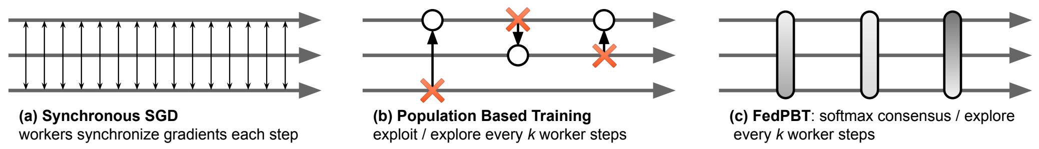
(a) Synchronous SGD
workers synchronize gradients each step
(b) Population Based Training
exploit / explore every k worker steps
(c) FedPBT: softmax consensus / explore
every k worker steps

Figure 1: SoftPBT leverages softmax policy consensus to keep population weights in sync so that it can utilize experience sharing 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 therefore cannot use experience sharing, and also Synchronous SGD, which averages gradients from each worker with uniform weighting and uses experience sharing.

duration. However, there is the possibility of doing better with a hybrid approach between large-scale synchronous training with experience sharing and PBT.

In this paper we introduce SoftPBT, such a hybrid algorithm that leverages softmax policy consensus to enable experience sharing between population members in PBT. While SoftPBT is applicable to supervised settings, for reinforcement learning in particular it enables order-of-magnitude sample efficiency improvements through *sample sharing*. SoftPBT is simple and works for any RL algorithm including on-policy algorithms. In our evaluation, we show that SoftPBT greatly outperforms random grid search, standard PBT, and HyperOpt baselines across a variety of RL tasks.

2 Background and Overview

Here we describe the relevant background of Deep Reinforcement Learning (Deep RL) and Population Based Training (PBT). We then overview our method, Softmax Policy Consensus Population Based Training (SoftPBT) 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 [18] and PPO [15] assumes that the samples are taken from the current policy being optimized. Off-policy algorithms such as DQN [13], 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 [13] and sample sharing. Employing experience replay (and therefore sample sharing) in on-policy approaches is an ongoing area of research [4].

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.

Softmax Policy Consensus Population Based Training: In distributed synchronous SGD (i.e., Figure 1(a)), multiple workers compute gradients over shared experiences, and synchronize gradients with each other at each step of the optimization process. This ensures all models are kept in sync so that gradients can be computed over all the experiences collected by the policies. However, this isn't the only way to synchronize the models. In federated learning [12], the gradients of each worker are weighted by dataset magnitude and averaged to ensure synchronization.

Table 1: Comparison of SoftPBT with standard Population Based Training, Synchronous SGD, and independent trial execution. Since RL training is often bottlenecked by sampling complexity in practice, SoftPBT 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,	$O(N \times T)$	$O(N \times T)$	none	any
Population Based Training	$O(N \times T)$	$O(N \times T)$	**schedules**	any
SoftPBT (ours)	$O(N \times T)$	$\mathbf{O(T)}$	**schedules**	RL training

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 policy consensus*, which takes a softmax over the objective value improvement with some temperature β.

Softmax policy consensus keeps the parameters of each worker closely in sync, only allowing drift between the synchronization intervals and ensuring that experience sharing does not damage training for on-policy algorithms. 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 softmax policy consensus, we minimize the efficiency lost by removing the PBT `exploit()` mechanism. 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 SoftPBT, our variant of PBT that utilizes *policy consensus* and itexperience sharing for significant efficiency gains. In contrast to distributed off-policy RL approaches that already naturally share experiences between multiple actor-learner pairs, *experience 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. *Policy consensus*, via softmax, ensures that a population of policies does not diverge greatly.

Experience sharing: By sharing experiences between population members, 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.

Policy consensus: 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 agents 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 experience sharing.

When $\beta \to 0$, we obtain a synchronous optimization process where weights are uniformly averaged. On the flip side, as $\beta \to \infty$, the procedure becomes similar to Population Based Training [6], where the weights of top-performing members (i.e., workers) of the population are copied to other workers. Softmax averaging is a more effective weight exploitation step than uniform averaging. As the results show, this uniform averaging does not achieve the same convergence as a single actor because the policies diverge significantly between perturbation intervals. 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 modifying PBT (Algorithm 1) with experience sharing and softmax policy consensus, we arrive at SoftPBT (Algorithm 2). 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 SoftPBT.

Algorithm 1 Population Based Training

```
 1: procedure TRAIN(P)                                          ▷ initial population P
 2:     for $(\theta, h, p, t) \in P$ (asynchronous in parallel) do
 3:         while not end of training do
 4:             $D \leftarrow$ sample$(\theta|h)$          ▷ each member generates batch of samples w/hparams h
 5:             $\theta \leftarrow$ step$(\theta|h, D)$          ▷ one step of optimization w/hparams h, samples D
 6:             $p \leftarrow$ episode reward mean$(D)$   ▷ fitness is smoothed episode mean reward of D
 7:             if ready(p, t, P) then
 8:                 $h', \theta' \leftarrow$ exploit$(h, \theta, p, P)$  ▷ use the rest of population to find better solution
 9:                 if $h \neq h'$ then
10:                     $h, \theta \leftarrow$ explore$(h', \theta', P)$          ▷ produce new hyperparameters h
11:                     $p \leftarrow$ eval$(\theta)$                              ▷ new model evaluation
12:                 update P with new $(\theta, h, p, t+1)$                  ▷ update population
13:     return $\theta$ with the highest $p$ in P
```

4 Evaluation

We seek to answer the following questions in our evaluation:

1. How does SoftPBT compare to other hyperparameter optimization strategies, namely PBT, HyperOpt, random search, and can it even outperform the fixed-schedule in terms of sample complexity?
2. Does on-policy SoftPBT perform well with only experience sharing, or is policy consensus necessary as well?
3. How sensitive is SoftPBT to the softmax temperature hyperparameter (i.e., did we just shift the search problem to this new hyperparameter), and to population size?

4.1 Experimental Setup

We evaluate SoftPBT 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 six gym environments, three Atari and three MuJoCo: `HalfCheetah-v2`, `Humanoid-v2`, `Hopper-v2`, `QbertNoFrameskip-v4`

Algorithm 2 Softmax Policy Consensus Population Based Training

```
 1: procedure TRAIN($P, \beta$)                              ▷ initial population P, temperature $\beta$
 2:     for $(\theta, h, p, t) \in P$ (synchronous in parallel) do
 3:         while not end of training do
 4:             $D_\theta \leftarrow$ sample($\theta|h$)              ▷ each agent samples batch of size $\frac{1}{P}$ w/hparams h
 5:             $D \leftarrow$ concat($D_\theta$)                     ▷ concatenate all population minibatches
 6:             $\theta \leftarrow$ step($\theta|h, D$)              ▷ one step of optimization w/hparams h, samples D
 7:             $p \leftarrow$ episode reward mean($D_\theta$) ▷ fitness is smoothed episode mean reward of $D_\theta$
 8:             if allready(p, t, P) then
 9:                 $\theta^{'} \leftarrow$ update($\beta, \theta, p, P$) ▷ come to consensus with population on better weights
10:                 $h^{'} \leftarrow$ exploit($h, \theta, p, P$)         ▷ use population to find better hyperparams
11:                 if $h \neq h^{'}$ then
12:                     $h, \theta \leftarrow$ explore($h^{'}, \theta^{'}, P$)     ▷ produce new hyperparameters h
13:                     $p \leftarrow$ eval($\theta$)                         ▷ new model evaluation
14:                 update P with new $(\theta, h, p, t+1)$                  ▷ update population
15:     return $\theta$                                                      ▷ global model
16: procedure UPDATE($\beta, \theta, p, P$)           ▷ weights and evaluation results of entire population
17:     denom $\leftarrow \sum_P \exp \beta \times p$
18:     for all $(\theta, p) \in P$ do
19:         $\theta^{'} \leftarrow \frac{\exp \beta \times p}{denom} \times \theta$               ▷ softmax-weighting
20:     return $\sum_P \theta^{'}$                                           ▷ consensus
```

`PongNoFrameskip-v4` and `BreakoutNoFrameskip-v4`. We implemented SoftPBT on top of Tune [11], and use the PPO implementation with default hyperparameters unless otherwise specified from RLlib [10] for parallel training. Each trial configuration is evaluated four times, and we report the median plotting the 25th and 75th quantiles in the shaded regions.

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 a random selection from the same spread of hyperparameters combinations. We jointly tune the learning rate, discount rate, and entropy coefficient over distributions of [1e-2, 5e-3, 1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 5e-6], [0.997, 0.995, 0.99, 0.98, 0.97, 0.95, 0.9, 0.85, 0.8], [0.001, 0.01, 0.0] respectively to closely mirror the original PBT paper, which tunes learning rate, entropy coefficient, and intrinsic reward cost (the last of which is not applicable here). 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 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 SoftPBT use a perturbation interval of 4-8 PPO training iterations depending on the task (targeting $\sim$120 seconds per perturbation), a population size of 5, and a resampling probability of 0.25. All SoftPBT trials are run with a temperature of 2.0.

Benchmarks: We benchmark SoftPBT against PBT and HyperOpt. 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 `Breakout`), showing the non-triviality of the setup and importance of schedule learning.

4.2 Comparison to Baselines

We compare SoftPBT to PBT, HyperOpt with early stopping, random grid search, and the best fixed schedule oracle as identified by HyperOpt/random search. We find that randomly initialized trials of SoftPBT are consistently able to beat other hyperparameter optimization techniques to competitive reward benchmarks by an order of magnitude, and reach higher final rewards. We plot the mean episode reward over samples.

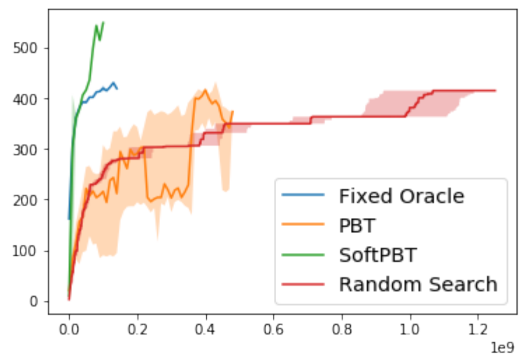

Figure 2: Breakout

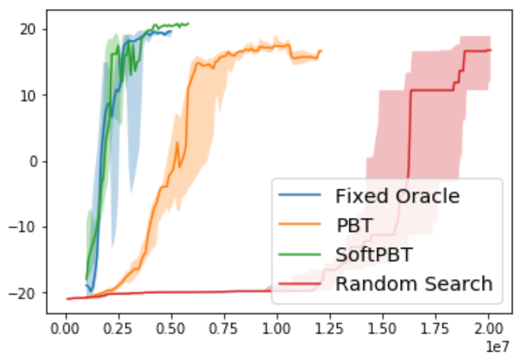

Figure 3: Pong

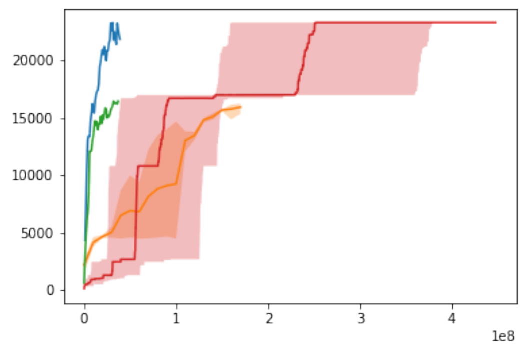

Figure 4: Qbert

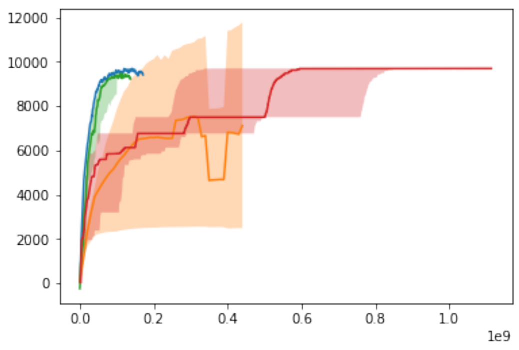

Figure 5: HalfCheetah

4.3 Impact of Proposed Techniques

In Table 2, we study the necessity of softmax policy consensus. First, we consider PBT with just experience sharing (PBT-ES). 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 policy consensus with SoftPBT stabilizes training and outperforms the other baselines. We find these results are general—SoftPBT requires both experience sharing and policy consensus 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 policy consensus. Our evaluation only considers the on-policy setting without importance correction, a strictly harder scenario.

4.4 Sensitivity to Hyperparameters

Finally, in Tables 3 and 4 we analyze the impact of the SoftPBT softmax temperature and PBT population size. We find that SoftPBT does reasonably well across a broad range of softmax

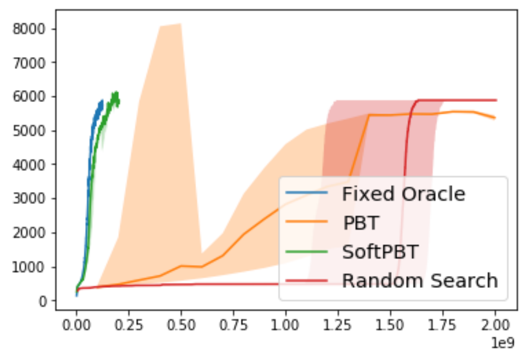

Figure 6: Humanoid

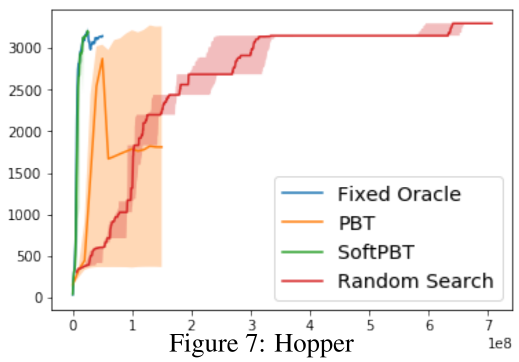

Figure 7: Hopper

Table 2: Ablation of policy consensus 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-ES	>100M	>100M	15 rew	45 rew
SoftPBT ($\beta = 4$)	**3M**	**6M**	**30 rew**	**140 rew**
SoftPBT ($\beta = 0$)	>100M	>100M	18 rew	55 rew

Table 3: Ablation of experience sharing (PBT-PC) 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-PC	80M	400M	600 rew	4800 rew
SoftPBT (N=4, $\beta = 0.0$)	21M	>400M	2500 rew	6500 rew
SoftPBT (N=4, $\beta = 0.5$)	21M	>400M	2700 rew	6500 rew
SoftPBT (N=4, $\beta = 1.0$)	19M	88M	2900 rew	8300 rew
SoftPBT (N=4, $\beta = 1.5$)	11M	**54M**	3500 rew	**9200 rew**
SoftPBT (N=4, $\beta = 2.0$)	19M	100M	2700 rew	8000 rew

temperatures, consistently achieving higher rewards than PBT at 100M timesteps. However, SoftPBT sometimes does not converge to the highest rewards if the softmax temperature is set too low, since the exploit step of SoftPBT becomes effectively disabled and the algorithm does not choose the best-performing weights. We also show that SoftPBT is robust across a range of population sizes, providing speedups for $N = 4$ to $N = 20$.

4.5 Summary

SoftPBT performs as well as or better than PBT on a range of problems, but critically the sample complexity of SoftPBT does not scale linearly with the population size. Full graphs with error bars can be found in the appendix; we note that all approaches have high variance. The ablation studies validate the importance of the techniques used in SoftPBT. SoftPBT without policy consensus (PBT-ES) performs poorly because the policies diverge too much to make experience sharing helpful. SoftPBT without experience sharing (PBT-PC) is not sample-efficient because experiences are not replicated.

Table 4: 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
SoftPBT (N=4)	93K	87.5 rew
SoftPBT (N=8)	112K	82.5 rew
SoftPBT (N=20)	**50K**	**90 rew**

SoftPBT 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 experience sharing, can potentially inhibit generalization to more complex tasks, since it collapses the population to one mode of behaviour. Nevertheless, we believe SoftPBT 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 [17, 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 [8, 9] have limited use when optimizing over hyperparameter schedules, which may have performance improvements only shown in later stages of training. SoftPBT 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 SoftPBT that simultaneously offers sample efficiency at the level of a single trial and the ability to also explore the hyperparameter space. We can upper bound the performance of other optimization methods which do not learn schedules, with the best grid search sample. As SoftPBT outperforms the best grid search sample, it would outperform other optimization methods which do not discover schedules. In comparison to portfolio-based and evolutionary strategies which do discover schedules, SoftPBT is 10x or more sample efficient for typical population sizes because it requires only the samples of a single trial to discover efficient schedules. The closest related work is HOOF [14]. With regards to HOOF, SoftPBT is more flexible in the sense that, like PBT, it can optimize arbitrary hyperparameters, not just those affecting the policy update directly. SoftPBT is also simpler in that it can treat the policy being optimized as a "black box", unlike HOOF's meta objective.

6 Conclusion

We present SoftPBT, an efficient variant of PBT that leverages softmax policy consensus to almost eliminate the sample complexity overhead incurred by maintaining a population of agents. We believe that SoftPBT is a superior alternative to PBT in many practical scenarios where efficient hyperparameter exploration is important. Our evaluation shows that SoftPBT 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] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

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

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

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

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

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

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

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

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

[17] 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.

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