Data Efficient Training for Reinforcement Learning with Adaptive Behavior Policy Sharing

Ge Liu
CSAIL, MIT
geliu@mit.edu

Heng-Tze Cheng
Google Brain
hengtze@google.com

Rui Wu
Google Brain
wrui@google.com

Jing Wang
Google Brain
jingconanwang@google.com

Jayden Ooi
Google Brain
jayden@google.com

Lihong Li
Google Brain
lihong@google.com

Ang Li
DeepMind
anglili@google.com

Wai Lok Sibon Li
DeepMind
sibon@google.com

Craig Boutilier
Google AI
cboutilier@google.com

Abstract

Deep Reinforcement Learning (RL) is proven powerful for decision making in simulated environments. However, training deep RL model is challenging in real world applications such as production-scale health-care or recommender systems because of the expensiveness of interaction and limitation of budget at deployment. One aspect of the data inefficiency comes from the expensive hyper-parameter tuning when optimizing deep neural networks. We propose Adaptive Behavior Policy Sharing (ABPS), a data-efficient training algorithm that allows sharing of experience collected by behavior policy that is adaptively selected from a pool of agents trained with an ensemble of hyper-parameters. We further extend ABPS to evolve hyper-parameters during training by hybridizing ABPS with an adapted version of Population Based Training (ABPS-PBT). We conduct experiments with multiple Atari games with up to 16 hyper-parameter/architecture setups. ABPS achieves superior overall performance, reduced variance on top 25% agents, and equivalent performance on the best agent compared to conventional hyper-parameter tuning with independent training, even though ABPS only requires the same number of environmental interactions as training a single agent. We also show that ABPS-PBT further improves the convergence speed and reduces the variance.

1 Introduction

Recent years have witnessed the success of deep reinforcement learning (RL) in solving complex sequential decision making problems such as games[Mnih et al., 2013, Schulman et al., 2015, Mnih et al., 2016]. However, it is yet proven to be effective in real world applications such as large-scale health-care or recommender system due to some practical constraints. First, there is no simulator available in many real-world problems and it is often expensive to do online interactions (e.g. through interacting with users). Second, only a small amount of online traffic and time budget is allocated for training new models. Last, the cost of deploying new models is too expensive for training to perform frequent behavior policy updates.

Optimization Foundations for Reinforcement Learning Workshop at NeurIPS 2019, Vancouver, Canada.

Hyper-parameter tuning and architecture design is key in the optimization of deep neural networks. However, unlike supervised learning, RL training requires subtle data-collection through interactions with the environment where traditional ways of hyper-parameter tuning with independent training shows their limits in terms of data efficiency. In this paper, we propose Adaptive Behavior Policy Sharing (ABPS), a data-efficient training algorithm for off-policy RL that allows sharing of experience collected by a single behavior policy that is adaptively selected from a pool of agents, which are trained with different hyper-parameters.

Our main contributions are as follows: 1) We revisit the algorithm selection for RL with a new aspect: improved exploration due to the ensemble bootstrapping effect, and experiment with different exploration strategies for algorithm selection to study their effect on deep RL optimization. 2) We propose a more practical setup where the switching of behavior policy is less frequent. 3) We evaluate the best and the overall performance of the learned target policies instead of only evaluating the regret (behavior policy reward), because in real world only a single best agent is deployed of which we care about its expected reward. We show that the best agent converges as effective as the agent trained with independent training in terms of performance, and the pool of agents gain higher overall return with reduced variance on top 25% agents. 4) We propose a novel adaptation of the Population Based Training (PBT) [Jaderberg et al., 2017, Li et al., 2019] and integrate PBT with ABPS. This variant has the potential to efficiently train and evolve hyper-parameters simultaneously so that better hyper-parameter selection is achieved. We observe faster convergence with even lower variance and higher median on the top 25% agents in the pool.

2 Related Work

Many studies have focused on improving data-efficiency through better exploration using distributional value functions. Osband et al. [2016] introduced a multi-head Deep Q network (DQN) where each head is trained with bootstrapped data. They showed that with a randomly selected policy from the ensemble of DQN heads the exploration can be improved. Other recent studies in this direction introduced quantile regression [Mavrin et al., 2019] and distributional RL [Bellemare et al., 2017]. These works provide better uncertainty estimates for the value function, however, they do not address the hyper-parameter tuning problem. Another strand of studies focus on utilizing the off-policy data. Jiang and Li [2015], Farajtabar et al. [2018] used doubly robust trick to improve off-policy evaluation of the state-action pairs. While this line of work has made the best use of off-policy data, there remains high bias and variance in the estimation inherited from the data.

Several studies related algorithm selection [Kotthoff, 2016] with reinforcement learning. Azar et al. [2013] showed that a bandit can choose the best RL policy with reasonable regret bound if a learned set of RL policies is given. However it does not taking the training and optimization process into account since the policies are fixed. Laroche and Feraud [2018] proposed a similar framework for optimization algorithm selection in RL, and provided theoretical analysis on the regret bound of behavior policy. However the RL algorithms are considered as black-box, and the paper has limited focus on how algorithm selection is affecting the exploration of RL and the performance of resulting target policies. In addition it adopted a setting which requires very frequent behavior policy selection.

3 Methods

3.1 General problem setup

We consider tasks in which an RL agent learns through sequential interactions with an environment E whose internal state is unobservable. At each time-step $t \in \mathbb{N}$, the agent selects an action from a legal action set $a(t) \in \mathcal{A}$ according to policy $\pi(t)$, and then receives from E a reward $r(t+1) \in \mathbb{R}$ and an observation $o(t+1)$ which is determined by the next state $s(t+1)$.

We assume the RL problem is episodic since it is a common set-up in games and real world problems, and policy selection on non-episodic RL is known to be hard [Azar et al., 2013]. To ensure the algorithm selection satisfies the "*leaning is fair*" assumption mentioned in [Laroche and Feraud, 2018], we mainly consider *off-policy* RL algorithms, even though the sharing of behavior policy can be adapted to *on-policy* algorithms with corrections (*e.g.*, importance sampling). For the purpose of demonstration, we further constrain our focus to Deep Q network (DQN) [Mnih et al., 2013], which uses a neural network function approximator with weights θ as a Q-network to estimate

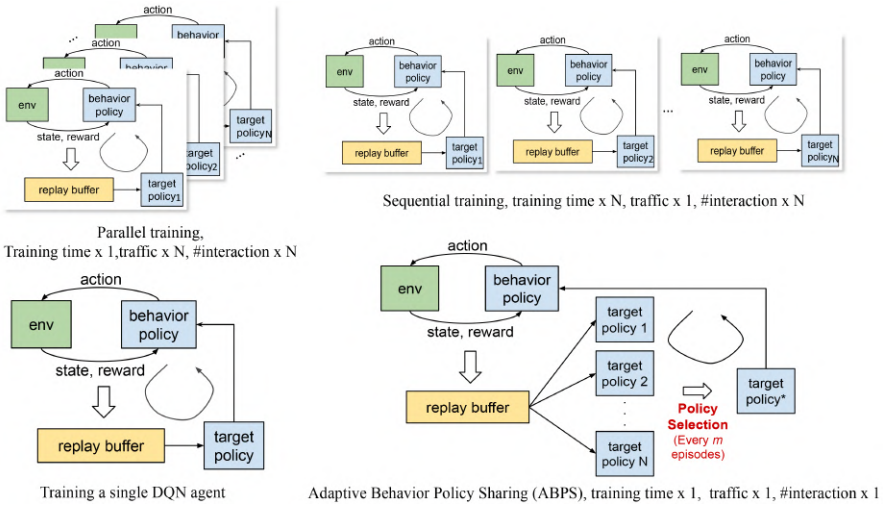

Figure 1: Illustration of conventional hyper-parameter tuning with sequential or parallel independent training and hyper-parameter tuning with Adaptive Behavior Policy Sharing.

the action-value function $Q(s, a; \theta)$, and utilizes a *replay buffer* $\mathcal{D}$ to store the agent's experiences $e_t =< s_t, a_t, r_t, s_{t+1} >$ at each time-step. It learns a greedy policy $a = \arg\max_a Q(s, a; \theta)$, but an ϵ-greedy policy of the Q is often used during training with annealing ϵ.

Training the Q-network involves first enumerating a candidate set of hyper-parameters, where a pool of network architectures and/or optimization hyper-parameters such as learning rate, ϵ decay period, etc., are selected. The goal is to find the best set such that the agent trained with that hyper-parameter set achieves best online evaluation results. As shown in Figure 1, conventional hyper-parameter tuning requires N times more interaction steps (where N is the size of hyper-parameter pool) as each agent collects experience on its own and a separate replay buffer is used. Moreover, it often requires either N times longer training time or more online traffic which is not desirable.

There are several more practical constraints that need to be taken into consideration in real world production scale systems:1) The cost of deploying new model is high, and usually serving a huge ensemble of models is not desirable. 2)Frequent change of online model may cause inconsistent service to the user.

3.2 Adaptive Behavior Policy Sharing (ABPS)

With the above mentioned constraints, we propose Adaptive Behavior Policy Sharing (ABPS) to improve the data-efficiency in hyper-parameter tuning by enabling sharing of experience among the agents while learning a *policy selection strategy* σ that selects behavior policy adaptively such that the value function networks are optimized efficiently. The overall work-flow of ABPS is illustrated in Figure 1. A pool $\mathcal{P}$ of K DQN action-value function networks $Q_1, \ldots, Q_K$ are trained simultaneously with a *shared* replay buffer $\mathcal{D}$, with each agent using a different hyper-parameter setup. During training, only *one* of the agents will be deployed as behavior agent at each time step. An ϵ-greedy policy of that agent is used to sample actions and the transitions are stored into the shared replay buffer using which all agents update their model parameters. After every m episodes, the received return is used to evaluate the corresponding behavior policy so that σ can be updated. Then a new behavior policy is chosen by σ for the next m episodes. This work-flow increases the stability of behavior policy during a period of m episodes and reduces the frequency of policy selection compared to [Laroche and Feraud, 2018]. However, it also increases the possibility of a bad behavior agent being chosen by chance which hurts the training of the rest by continuously collecting bad transitions. The effect of the policy selection period is studied in Section 4.1.

Learning the strategy σ is another exploration and exploitation trade-off problem. Inspired by [Osband et al., 2016] where uniformly sampled Q-function from an ensemble of randomized Q-networks could effectively result in better exploration, we experiment with a fully random policy selection strategy σ_{random} (naming it ABPS-random). Another natural selection for σ is multi-armed banditBubeck and Cesa-Bianchi [2012], which has been used in many algorithm selectionGagliolo and Schmidhuber [2006, 2008] and meta-learningFialho et al. [2010] problems. An UCB1 version of the bandit is often

used out of the principle of optimism in the face of uncertainty. Here we study additional exploration strategies for the bandit problem and propose three variations of ABPS-bandit:1) *ABPS-bandit-UCB*, where the arm with largest UCB value is chosen. 2) *ABPS-bandit-ϵ*, where with probability ϵ we randomly pick one agent from the pool and otherwise pick the arm with largest averaged reward. 3) textitABPS-bandit-softmax, where the agents are sampled from soft-max probability of the averaged reward of each arm.

At the end of the training, an ensemble of agents is obtained. Among them, the one or more top-performed agents are chosen to be deployed for serving. Note that the training only takes the same amount of time and environmental interaction as a single agent training run, whereas a naive hyper-parameter tuning requires up to K times more interactions and becomes less efficient. A full description of ABPS algorithm can be found in Algorithm 1. To account for the non-stationarity in deep neural network training, we used a bandit with sliding window instead. Another alternative is using a discount rate for historical values.

Algorithm 1 ABPS-DQN training procedure

Initialize DQN action-value function networks Q_1,Q_2,...,Q_K, shared replay buffer $\mathcal{D}$,bandit $\mathcal{B}$ with K arms
for t = 1,. . . , K **do**
  Run initial online evaluation for Q_t for n episodes, update arm t of $\mathcal{B}$ with the averaged reward $\bar{X}_t = \bar{R}_t$
repeat
  if ABPS_TYPE = `random` **then** Randomly select one network as behavior Q^* function
  else if ABPS_TYPE = `soft-max` **then** Sample behavior Q^* function with $p(Q_i) = \frac{e^{\bar{X}_i}}{\sum_{j=1}^{K} e^{\bar{X}_j}}$
  else if ABPS_TYPE = ϵ-`greedy` **then**
    With probability ϵ randomly select one network as behavior Q^* function
    Otherwise select Q_{I_t} as behavior Q^* function where $I_t = \arg\max_{0 \leqslant i \leqslant K} \bar{X}_i$
  else if ABPS_TYPE = `UCB` **then**
    select Q_{I_t} as behavior Q^* function where $I_t = \arg\max_{0 \leqslant i \leqslant K} \bar{X}_i + \sqrt{\xi \log t / n_i}$
  for episode = 1,. . . ,m **do**
    repeat
      Roll out s steps with $\pi(Q^*)$ and store transitions in $\mathcal{D}$
      Train Q_1,. . . ,Q_K for 1 step
    until episode ends
  Update arm I_t of $\mathcal{B}$ with averaged m-episode reward $\bar{R}_{I_t}$: $\bar{X_{I_t}} \leftarrow \frac{n_{I_t}\bar{X}_{I_t} + \bar{R}_{I_t}}{n_{I_t}+1}$, $n_{I_t} \leftarrow n_{I_t} + 1$
  $t \leftarrow t + 1$
until iteration limit reached

3.3 Adaptive Behavior Policy Sharing with evolving hyper-parameters

Given the complexity of the hyper-parameter and neural architecture space, it is possible that the true global optimal is not covered by the empirically chosen pool of hyper-parameters. The optimal hyper-parameters are also highly non-stationary. Recent studies have proposed techniques to conquer such deficiency, one of which is called Population Based Training (PBT)[Jaderberg et al., 2017, Li et al., 2019]. The idea is to use information from the rest of the population to refine the hyper-parameters through periodically substituting bad workers with promising workers (exploit) and random evolution of hyper-parameters (explore). However, each worker in the pool is trained independently, which again requires K times interactions with the environment where K is the number of workers. Given that both PBT and ABPS utilizes sharing of information across the population and deals with an exploitation/exploration problem, we propose a hybrid of the two, named ABPS-PBT, such that both data efficiency and hyper-parameter search efficiency can be achieved.

Figure 2 shows the overall work-flow of ABPS-PBT. Some adaptations are required to make PBT compatible with ABPS. The vanilla PBT exploitation uses real-time online evaluation for all workers in the pool, while in ABPS-PBT the evaluation is replaced by the ABPS-bandit value tables. Even though the sliding window bandit keeps relatively up-to-date values for each arm, we set an additional threshold τ_{update} on the last update time to detect outdated values. An online evaluation will be triggered immediately when the arm is not sufficiently up-to-date for PBT to do useful exploitation. Both PBT and ABPS has a period set between the triggering of algorithms. We set the period for PBT to be a multiples of the period for ABPS, since ABPS's policy selection happens more frequently than PBT-exploitation. In addition to model parameters and hyper-parameters, the bad worker's bandit value is also substituted by the counterparts of the good worker during PBT-exploitation. A detailed description of the algorithm can be found in Algorithm 2.

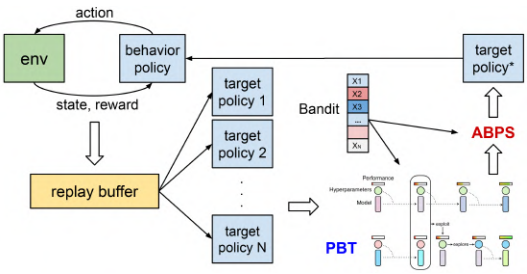

Figure 2: Illustration of ABPS combined with Population Based Training

Algorithm 2 ABPS-PBT-DQN training procedure

Initialize population pool $\mathcal{P}$ of K DQN action-value function networks $Q_1,\ldots,Q_K$ associated with K sets of hyper-parameters $h_1,h_2,...,h_K$, initialize shared replay buffer $\mathcal{D}$,bandit $\mathcal{B}$
repeat
  if `pbt-ready` **then**
    for each agent Q_i **do**
      if most recent update time of arm $i > \tau_{update}$ **then** run $\texttt{eval}(Q_i)$ and update $\bar{X}_i$
      Exploit using the bandit values: $h_i', \theta_i', \bar{X}_i{}' \leftarrow \texttt{exploit}(h_i, \theta_i, \bar{X}_i, \mathcal{P})$
      if $\theta_i \neq \theta_i'$ **then** $h_i, \theta_i \leftarrow \texttt{explore}(h_i', \theta_i', \mathcal{P})$
  Select behavior policy $\pi(Q^*)$ with ABPS (described in Algorithm 1)
  for episode = 1,...,m **do**
    repeat
      Roll out s steps with $\pi(Q^*)$ and store transitions in $\mathcal{D}$, train $Q_1,\ldots,Q_K$ for 1 step
    until episode ends
  Update the mean value of the chosen arm with averaged episode reward (see Algorithm 1)
  $t \leftarrow t + 1$
until iteration limit reached

3.4 Algorithm evaluation

Instead of examining the behavior policy reward collected during training (*i.e.*, the regret of the behavior policy), we run a separate online evaluation for 50 episodes for every agent at each training epoch. The online evaluation reward reflects the goodness of the individual agent in the pool. We choose the agent with the best online evaluation reward at the end of training, and compare it with the best agent achieved from full hyper-parameter tuning without ABPS. This simulates the real world scenario where we care about not only the regret during training but also the serving performance. To evaluate the overall performance of the ensemble of the agents, we also look at the top 25% quantile and the variance of the rewards.

4 Experiments

4.1 DQN with a large hyper-parameter pool

We show effectiveness of ABPS across 3 Atari games (supplement figure 1) with a pool of 4 different architectures (See appendix 1). We further test ABPS on large hyper-parameter pool, where a pool of 16 agents is drawn from a prior distribution, where the marginal probability over the four architectures is $[0.2, 0.2, 0.2, 0.4]$ for normal, wide, deep and small correspondingly, and a random perturbation (scaled by $0.9 \sim 1.1$) is made to the number of convolutional and fully connected units. The learning rate λ and ϵ decay period is log-uniformly sampled from $[0.00001, 0.005]$ and $[2.5e5, 4e6]$ respectively. Figure 3 shows the online evaluation performances of best agent and top 25% quantile through out the training, where all ABPS variants achieved equivalently good convergence as the best hyper-parameter in the pool and ABPS-UCB convergence the fastest. The best agents selected by ABPS-bandit methods are equivalent to the best in independently trained agents, showing ABPS's capability to recover the traditional hyper-parameter tuning results (if not better) with almost no computation overhead and much fewer environmental interactions. We also discovered ABPS's capability of improving the overall performance of the ensemble, with higher top 25% quantile and lower variance. Figure 4 shows the accumulated frequency of agents with different architectures and optimization hyper-parameters being chosen as the behavior policy for ABPS. We observed an increasing frequency of using wider networks against the others, and the adaptation of hyper-parameters. Larger learning rate and high ϵ are preferred on the earlier training rounds and are both reduced at later training rounds, which is consistent with the annealing strategy in prior studies on the optimization of deep RL.

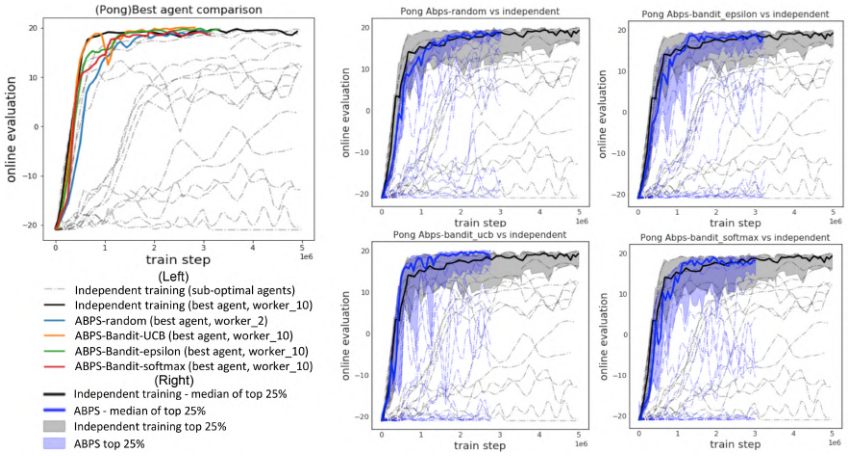

Figure 3: Best agent online evaluation (left) and top 25% quantile (right) of the online evaluation performance of the whole ensemble of hyper-parameter pools. Black line represents the best agent achieved by traditional hyper-parameter tuning without ABPS and dashed grey lines are the sub-optimal agents in the hyper-parameter pool trained without ABPS.

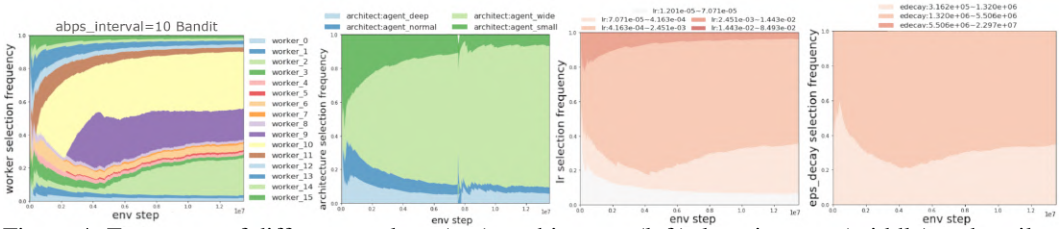

Figure 4: Frequency of different workers (top), architecture (left), learning rate (middle) and epsilon decay speed (right) being chosen as behavior policy at each time step.

We also study the effect of policy selection period on ABPS performance and the result is shown in appendix figure 2. We found that ABPS variations that rely on random exploration (random, ϵ-greedy) are more affected by longer selection period than value based exploration (UCB, softmax). With up to 60 episodes of ABPS interval, the ABPS-Bandit-UCB still achieves comparable performance with traditional hyper-parameter tuning.

4.2 Integrating ABPS with Population Based Training

With the same experimental setup as section 4.1, we evaluate the performance of ABPS-PBT. As shown by Figure 5, the best agent convergence speed is dramatically improved after combining with PBT, and the top 25% agents are significantly improved with much smaller variance and higher median, showing the effectiveness of ABPS-PBT.

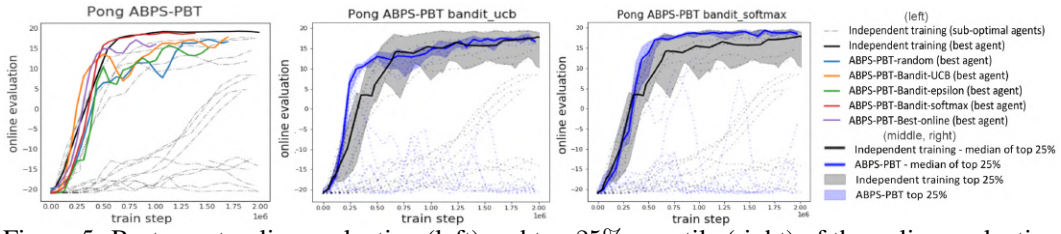

Figure 5: Best agent online evaluation (left) and top 25% quantile (right) of the online evaluation performance of the whole ensemble of hyper-parameter pools using ABPS-PBT.

5 Conclusion

We propose Adaptive Behavior Policy Sharing, a data-efficient training algorithm that allows sharing of experience collected by adaptively selected behavior policies. We further adapt Population Based Training and proved that it is compatible with ABPS. We experimented with multiple Atari games with up to 16 hyper-parameter/architecture setups, and achieved superior overall performance, smaller variance on top 25% agents, and matching performance on the best agent compared to conventional hyper-parameter tuning with independent training, even though ABPS only needs the same number of environmental interactions as training a single agent. We also show that ABPS-PBT has the potential to improve both the convergence speed and the variance.

References

Mohammad Gheshlaghi Azar, Alessandro Lazaric, and Emma Brunskill. Regret bounds for reinforcement learning with policy advice. In *Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECMLPKDD'13, pages 97–112, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-40987-5. doi: 10.1007/978-3-642-40988-2_7. URL https://doi.org/10.1007/978-3-642-40988-2_7.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL http://arxiv.org/abs/1707.06887.

Sébastien Bubeck and Nicolò Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012. doi: 10.1561/2200000024. URL https://doi.org/10.1561/2200000024.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. *CoRR*, abs/1802.03493, 2018. URL http://arxiv.org/abs/1802.03493.

Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Ann. Math. Artif. Intell.*, 60(1-2):25–64, 2010. doi: 10.1007/s10472-010-9213-y. URL https://doi.org/10.1007/s10472-010-9213-y.

Matteo Gagliolo and Jürgen Schmidhuber. Learning dynamic algorithm portfolios. *Ann. Math. Artif. Intell.*, 47(3-4):295–328, 2006. doi: 10.1007/s10472-006-9036-z. URL https://doi.org/10.1007/s10472-006-9036-z.

Matteo Gagliolo and Jürgen Schmidhuber. Algorithm selection as a bandit problem with unbounded losses. *CoRR*, abs/0807.1494, 2008. URL http://arxiv.org/abs/0807.1494.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Nan Jiang and Lihong Li. Doubly robust off-policy evaluation for reinforcement learning. *CoRR*, abs/1511.03722, 2015. URL http://arxiv.org/abs/1511.03722.

Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pages 149–190. Springer, 2016.

Romain Laroche and Raphael Feraud. Reinforcement learning algorithm selection. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyoDInJ0-.

Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. A generalized framework for population based training. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1791–1799, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330649. URL http://doi.acm.org/10.1145/3292500.3330649.

Borislav Mavrin, Shangtong Zhang, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration. *CoRR*, abs/1905.06125, 2019. URL http://arxiv.org/abs/1905.06125.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. URL https://arxiv.org/pdf/1312.5602.pdf.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn.pdf.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

Appendix 1

DQN with a small hyper-parameter pool

We first experiment with a small hyper-parameter pool with 4 candidate network architectures:

- Agent normal(Mnih et al. [2013]): (16, (8, 8)) -> (32, (4, 4)) -> 256
- Agent small: (8, (8, 8)) -> (8, (4, 4)) -> 32
- Agent wide: (32, (5, 5)) -> (64, (3, 3)) -> 1024
- Agent deep: (32,(8, 8)) -> (64, (4, 4)) -> (64, (3, 3)) -> 512

Where numbers in parenthesis represent the number of convolutional filters and their corresponding shapes, and numbers on the most right represent the size of fully connected layer. We trained an ensemble of 4 agents with each using one of the candidate architectures on Atari Pong and Breakout, and an ensemble of 8 agents with 6 of them using variation of small architectures on Boxing to test the effect of bad agents on the different ABPS algorithms. Figure **??** shows the best agent performance of 3 ABPS variations and the performance of all agents trained independently without ABPS. ABPS-bandit methods achieved better evaluation performance on all three games with almost equivalent convergence speed with the independent training of the best agent in the pool. ABPS-Bandit-UCB converges better and faster than ABPS-Bandit-ϵ on Pong and Boxing. Surprisingly, even a random policy selection strategy could result in the same level of performance as ABPS-Bandit, demonstrating the power of exploration through ensemble bootstrapping effect. However, when the hyper-parameter pool is dominated with sub-optimal choices, the random algorithm is largely affected by bad agents, resulting in a drastic drop of best agent performance.

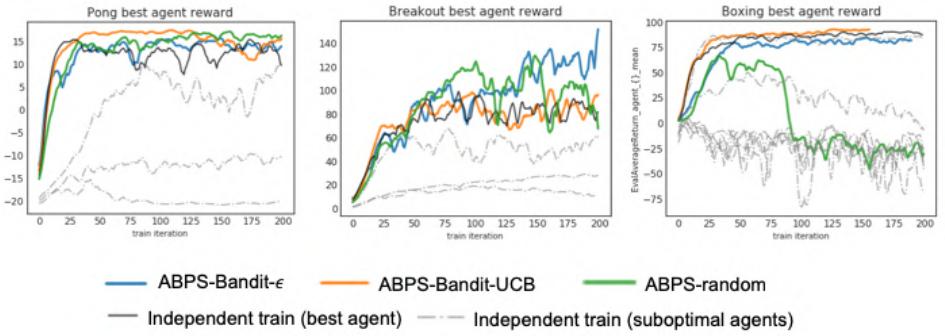

Figure 1. Best agent online evaluation performance on Atari Pong (left), Breakout (middle) and Boxing (right) for hyper-parameter pool of size 4 (left, middle) and 8 (right, 6 out of 8 are small networks). Black line represents the best agent achieved by traditional hyper-parameter tuning without ABPS and dashed grey lines are the sub-optimal agents in the hyper-parameter pool trained without ABPS.

Effect of policy selection period length

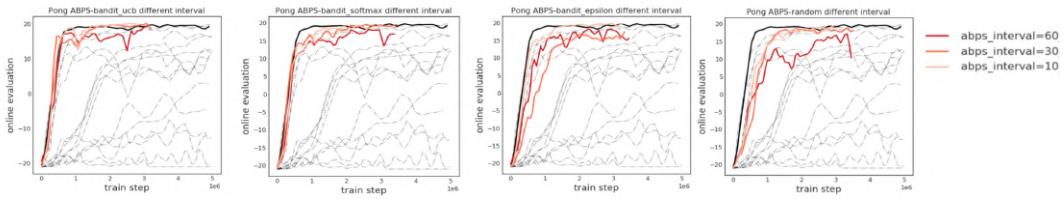

Figure 2. Effect of policy selection period length on the ABPS training using different exploration strategies (UCB, softmax, ϵ-greedy and random from left to right). ABPS variations that rely on value based exploration (UCB, softmax) are less effected than strategies that rely on random exploration (random, ϵ-greedy)