Automated Dynamics Curriculums for Deep Reinforcement Learning

Sean Metzger
UC Berkeley, UC San Francisco
sean.metzger@berkeley.edu

Abstract

Humans often make the dynamics of a task easier (e.g. using training wheels on a bicycle or a large voluminous surfboard) when first learning a skill before tackling the full task with more difficult dynamics (riding a bike without training wheels, surfing a smaller board). This can be thought of as a form of curriculum learning. However, this is not the paradigm currently used for training agents using reinforcement learning (RL). In many cases, agents are thrown into the final environment, and must learn a policy from scratch in the context of the final dynamics. While previous work on curriculum learning for deep RL has sought to address this problem by changing the tasks agents are solving, or the starting position of the agent, no work has derived a curriculum by modifying the dynamics of the final environment. Here, we study using *assist* - simplifying task dynamics - to accelerate and improve the learning process for RL agents. First, we modify the physics of the `LunarLander-v2` and `FetchReach-v1` environments to allow us to adjust the amount of assist provided with a single parameter α, which scales the amount which an agent is nudged and hence assisted towards a known end goal during training. We then show that we can automatically learn schedules for assist using a population based training approach that results in faster agent convergence on the evaluation environment without any assist, and better performance across continuous control tasks using state of the art policy gradient algorithms (proximal policy optimization). We show that our method can also scale to off-policy methods such as Deep Deterministic Policy Gradients. Furthermore, we show that for tasks with sparse rewards, assist is critical to agent learning as it allows exploration of high-reward areas and use of algorithms that fail to learn the task without assist. We also uncover that population based tuning approaches stabilize training of policy gradients *without* tuning of any additional hyperparameters.

1 Introduction

Reinforcement Learning (RL) is a powerful method for training agents to optimize reward functions on a variety of complex tasks, and has been demonstrated on tasks such as Atari games [1] and robotic manipulation tasks [2]. However, agents can be slow to learn, requiring hundreds of sampled trajectories for training data, which presents challenges for using RL to train agents in situations when sampling is expensive or when situations require rapid agent learning. Furthermore, tasks with sparse rewards are challenging to train - and sometimes training does not work at all if the agent is unable to achieve the sparse reward frequently.

One tactic for expediting learning that is popular for teaching humans new tasks is the use of assist. Here we define assist as an extrinsic force pushing one towards a goal. Assist could be thought of as the hand of a teacher guiding a student through a tennis swing, or the forces of training wheels keeping a bike upright. Another example comes from the Brain-Computer Interface literature [3, 4],

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

where a human learns to control a cursor using their neural activity in a closed-loop setting. At first the cursor is extrinsically guided to the desired position, before the amount of assist is tapered down until the human is completely in control of the brain-computer interface. Could such a strategy be effective for RL?

This paper seeks to answer the following questions: can assist be used for training reinforcement learning algorithms? Would making task dynamics easier or harder allow agents to learn meaningful policies? Would policies learned with easier dynamics be functional once the original dynamics are reintroduced? How can we determine functional assist schedules?

This work answers those questions by implementing assist mechanisms into two control tasks for reinforcement learning - guiding a lunar lander, and simulated robotic locomotion. We show that this can expedite learning and lead to improved convergence speed and reward values on two control tasks. The main contributions of this work can be summarized as follows:

- Formalizing and implementing assist for goal-oriented control tasks
- A novel and practical approach for automatically finding assist schedules
- Empirical evaluations showing that this approach can solve robotic manipulation and control tasks with higher convergence speed than using the same algorithms without modulating assist.

2 Previous Work

2.1 Curriculum Learning

This work seeks to implement an assist curriculum, hence it is important to review the curriculum learning literature.

The earliest example of curriculum learning in modern deep learning comes from [5]. A neural network was first presented with examples that a simple classifier was very sure of, and slowly trained on examples that simpler classifiers found more difficult to classify. This resulted in improved performance over training on samples in random order. However the idea of training models on simpler tasks before graduating to harder tasks was first proposed by [6], who trained neural networks on a restricted set of simple data for learning grammars before increasing the complexity. Doing so was shown to be critical to learning.

It is also possible to think of the extremely popular process of finetuning and pretraining neural networks (e.g. [7]) as a form of curriculum learning.

2.2 Curriculum Learning for Deep Reinforcement Learning

Previous work in curriculum learning [5] in RL has focused on changing the task itself from easy to hard [8], changing initial states from close to the reward to farther and farther away [9], and progressively changing the goals that the RL agent is trying to meet [10]. Another approach, taken by Mix&Match [11] is to learn a curriculum over agents by effectively bootstrapping solutions found by simpler agents for more complex tasks.

To date, no work has focused on using a curriculum of *dynamics* as we do here.

2.3 Hyperparameter Scheduling

An ideal assist curriculum will modulate the amount of assist that is provided to an agent throughout the learning process. For example, one could imagine beginning learning with a high assist amount, then reducing the amount of assist throughout training. Ideally, we would be able to automatically search for this assist schedule.

Learning hyperparameter schedules has long been an important field of study for deep learning, where learning rate schedules have been shown to result in improved performance for many tasks (e.g. [7]). Within deep Reinforcement Learning, finding a *schedule* for hyperparameters has been shown to improve performance on a wide range of tasks, including DeepMind-Lab, StarCraft II and Atari [12]. This schedule is derived through population based training, which is based on exploration

and exploitation of hyperparameters. In population based training, a set of models explores a set of hyperparameters in parallel. After performance is evaluated, then population based training keeps models with the best performing hyperparameters, clones them, then adjusts the hyperparameters of cloned models to explore new spaces.

Moreover, recent work [13] has shown that it is possible to learn a schedule of augmentations to improve performance of image classifiers on various datasets. The work uncovered augmentation schedules, where augmentations early in training used low augmentation strengths, before introducing higher augmentation strengths later in training - hence effectively deriving a *curriculum* of augmentations for training classifiers.

3 Assisting RL agents

3.1 Training agents

We want to derive a policy π, parameterized by a vector θ, π_θ to maximize the objective function:

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] \tag{1}$$

As shown in [14], we can rewrite this as

$$E_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

Where $\hat{A}_t$ is the estimated advantage.

We use Proximal Policy Optimization (PPO) [14] as our main algorithm to optimize equation 2 . PPO is a policy gradient method that seeks to optimize the following modified version of equation 2 to ensure that the policy's deviation from the previous policy is relatively small.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{3}$$

Here θ is the policy parameter, $\hat{E}_t$ is the empirical expectation over timesteps, r_t is the ratio of probability under the new and old policies, $\hat{A}_t$ is the estimated advantage at time t, and ϵ is a hyperparameter. This objective essentially implements a way to do a trust region update [15] that is compatible with stochastic gradient descent.

3.2 Assist approach

3.2.1 LunarLander-v2

We seek to assist the agent by making the dynamics of the environment easier. Essentially, this means 'guiding' the agent towards their goal or towards a high reward state. This can take many forms, and we explore two variants thereof in this work.

Firstly, for `LunarLander-v2` we directly modify the lander's velocity at each timestep. Taking inspiration from the brain-computer interface literature [3], we assume that an optimal agent would be trying to move towards the goal state at each timepoint, and adjust the velocity v_t at each timepoint to make the dynamics easier:

$$v_t = \alpha v_{assist,t} + v_{original,t} \tag{4}$$

Here $v_{original,t}$ corresponds to the true velocity of the lunar lander at time t before the assist velocity is applied. $v_{assist,t}$ is the assist velocity at time t which is derived as follows:

$$v_{assist,t} = \frac{x_{goal} - x_{curr}}{dt} \tag{5}$$

Here x is a 2-d vector with the current position of the agent, and dt is the frequency at which the environment operates. x_{curr} is the current position of the lander, and x_{goal} denotes the goal or high reward position we want the lander to end in.

We apply the assist velocity as an impulse using the `Box2D` physics engine.

If the assist level α is set high, the lander will be strongly guided towards the goal state no matter the actual velocity used.

3.2.2 FetchReach-v1

For the FetchReach environment, we use the FetchReach-v1 task, where we train a robotic arm to reach a target endpoint. The action space for the Fetch robotics environment is given as a four dimensional vector $(\delta_x, \delta_y, \delta_z, o)$. The first three dimensions of the vector correspond to the desired change along the axes x, y, and z for the gripper of a robotic arm. The final element in the action vector, o corresponds to the gripper being open or closed and is not changed with assist. The fetch robot then abstracts these desired changes in movement into an actual position. Hence by modifying each δ with an assistive δ, we achieve the same result as in equation 5.

Explicitly, we compose the assisted action as

$$(\delta_x, \delta_y, \delta_z, o) = (\delta_{x,original}, \delta_{y,original}, \delta_{z,original}, o_{original}) + \alpha(\delta_{x,assist}, \delta_{y,assist}, \delta_{z,assist}, 0) \tag{6}$$

We define

$$\delta_{x,assist} = x_{goal} - x_{curr} \tag{7}$$

Where x_{curr} is the current location, x_{goal} is the desired or high reward location, and so on for y, z.

It is worth noting that the Fetch robotics environment automatically scales the whole action vector by $.05$ to limit huge and impossible movements, providing a second layer of assist scaling beyond our scaling with α.

3.3 Finding schedules for α using population based training

Critical to the success of our method and it's ability to scale is finding a curriculum for α automatically on a variety of tasks. While increased assist will help agents recover high rewards during parts of training, we assume that it will need to be lowered so that the agent learns good control in the environment without assist where we want it to operate. However, it is unclear how to adjust α, and hand-tuning schedules for α would require hundreds of runs for each environment we want to learn an assist function for. Hence we used Population Based Training (PBT) to automatically discover assist schedules. PBT has been used to find augmentation [13] curriculums and learning rate schedules [16] and is thus a natural choice for us. We describe the formulation of the search in the format of PBT experiments as in [16, 13].

Step: In each iteration we train the policy for 1 iteration in an environment with assist level α.

Eval: We evaluate the policy on the environment without any assist by evaluating the mean reward over 10 episodes. This is so that we are moving towards policies that achieve the best performance on the final evaluation environment.

Ready: A trial is ready to go through the explore-and-exploit process when 1 PPO iteration has elapsed.

Exploit: We use truncation selection as in [16] where a trial in the bottom quarter of the population clones the weights and hyperparameter α of a model in the top 25%.

Explore: When a new trial is cloned from a successful trial, we either resample α from a distribution of possible values (we use a loguniform distribution) 25% of the time, or perturb the original value to be either .8 to 1.2 times its original value with equal probability of each of the two values being selected as in [16].

4 Experiments & Results

For all experiments we use `Ray` [17]. We take advantage of Ray's Population Based Training implementation, as well as their implementation of PPO and Deep Deterministic Policy Gradients

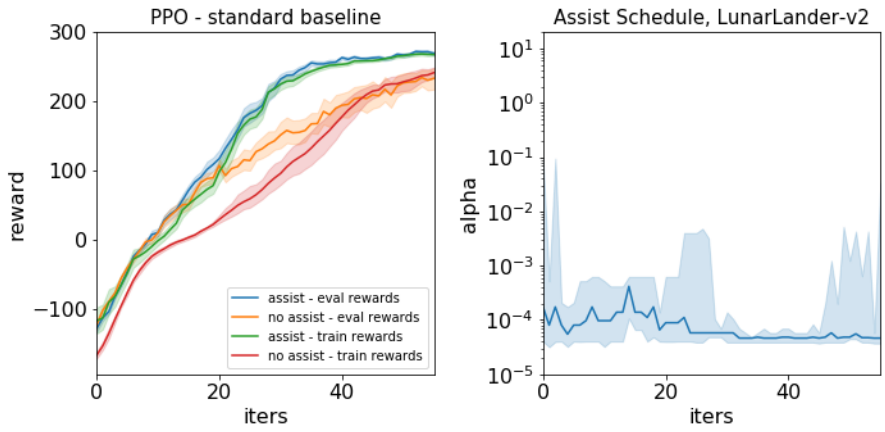

Figure 1: Evaluation of PPO with and without an assist schedule in the LunarLander-v2 environment. Left plot shows rewards vs the number of iterations. Thick line indicates mean of top half of agents in the population, shaded region indicates bootstrapped 95% confidence intervals. In all figures, eval rewards are evaluated in the environment without any assist, wheras train rewards are evaluated with assist on (if applicable). Right plot, assist schedule for one run with assist. Thick line indicates median assist amount for top half of agents in the population, shaded region shows boostrapped 95% confidence interval. Assist helps the agents learn rewards more quickly over traditional PPO training. The reward schedule shows that a small amount of assist earlier in training helps the agent achieve faster convergence and obtain better final performance. The assist later in training goes up a little bit, likely due primarily to exploration.

(DDPG). We use the LunarLander-v2 and FetchReach-v1 environments from OpenAI's Gym toolkit [18].

4.1 LunarLander-v2

We first evaluate the LunarLander environment, since it is a goal based control task that has been studied in the context of assisting agent learning (by inferring goals) in previous work [19]. We use PPO with a population size of 16. We initialize models with an assist level from a loguniform distribution from 10^{-5} to 1. During exploration, we resample from a loguniform distribution from 10^{-4} to 1 or scale values by .8 or 1.2.

We compare 3 runs with a population of 16 agents trained using PPO both with without using assist. All plots show the top half of performers in their populations, since recently perturbed agents may have bad performance. Throughout our experiments, the only hyperparameter tuned during PBT is α.

As shown in figure 1 using an assistive approach hugely improves performance over traditional PPO without assist. We also see that we are able to automatically learn an assist schedule that starts with a small α, before increasing α around 10-20 iters then reducing it once more near the end of training.

Because it was initially unclear if the improved performance from our assist method was due to the assist process itself, or just the advantage presented by PBT of throwing out bad models and replacing them with clones of good models early in training, given the challenges of training with policy gradients. Hence, we compared our assist approach to using PBT's exploitation with α constantly set to 0 in figure 2. This showed improved results without assist, but models with assist were still able to outperform models without assist, and reached higher rewards more quickly. Nevertheless, this demonstrates that using a population of models and exploiting the best model leads to far better results when using policy gradients.

4.1.1 Ablations

To more closely examine the importance of a schedule and the effects of using different models of assist, we trained models without using any schedule, and then using a naive schedule where assist

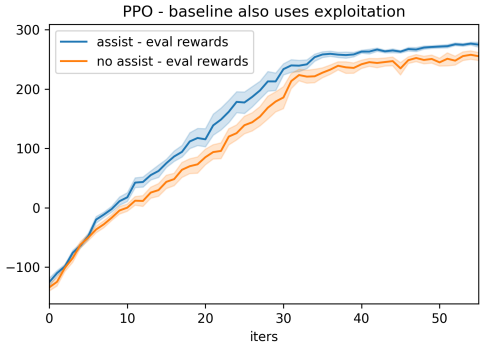

Figure 2: PPO - with Population Based Training exploitation applied to the baseline without the modulation of any hyperparameters: For a more fair comparison, we compare our assist with a population of models with no assist that where the bottom quarter of models are replaced by PBT every iteration. Because bad models are eliminated throughout training, population based training actually improves training significantly for high variance policy gradient techniques, even without actually tuning any hyperparameters. Nevertheless, our assist method is still able to beat the baseline and achieves stronger performance in fewer iterations.

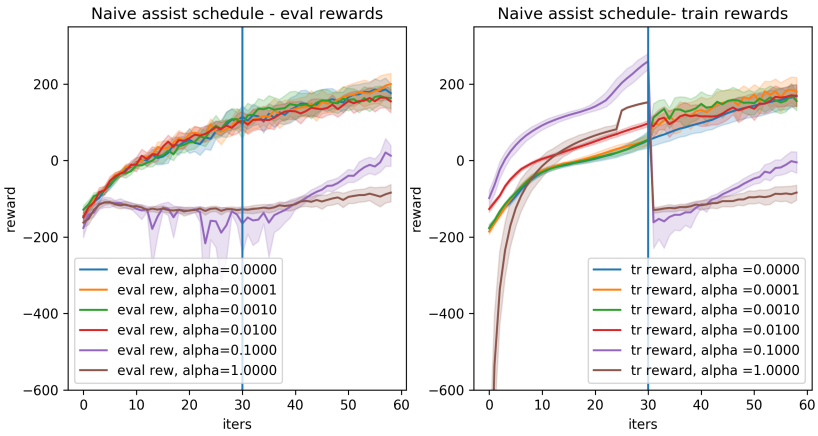

Figure 3: Training curves across 16 runs of PPO on LunarLander-v2 when using a naive assist schedule, where we shut off assist halfway through training at iteration 30, denoted by the vertical blue line. Despite performing better in environments with the assist on, policies trained with high assist start doing poorly when assist is turned off and must essentially start from scratch. Policies trained with smaller amounts of assist perform comparably to policies trained without assist and continue to learn and perform well even after assist is shut off.

was simply turned off halfway through training, and no exploitation or exploitation was done. Results are shown in Figure 3 and 4.

Figure 3 serves a sanity check that assist works as intended. Indeed, with increasing assist we see better rewards on the assisted environment earlier in training. However, using a high assist of 1 surprisingly does worse than using a smaller amount of assist (.1). This is likely because the assist amount was too high and resulted in overly jerky lander movements that were hard to correct. It is notable that policies with high levels of assist ($\geq$ .1) do not generalize well to the environment with no assist. This emphasizes that it is important to keep assist levels relatively small - otherwise the agent is essentially optimizing its policy within a completely different environment.

Both figure 3 and 4 emphasize the importance of a good assist schedule. Simply leaving assist on does not account for the performance improvements we saw in figures 1 and 2

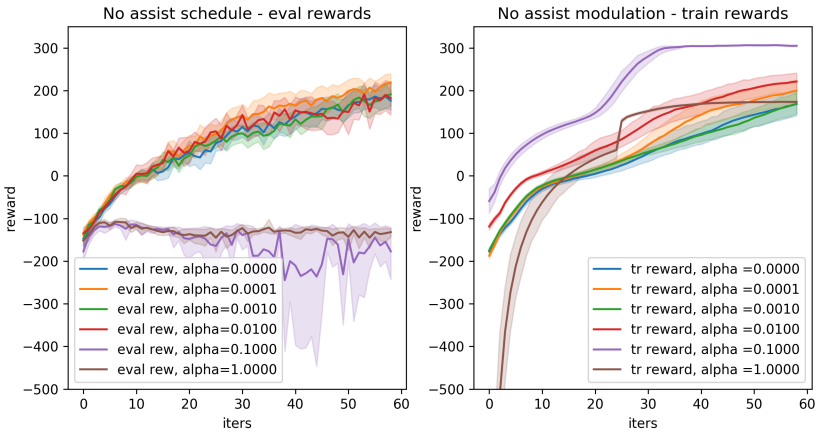

Figure 4: Evaluation of 16 runs with no assist modulation (assist is held constant throughout the run) on LunarLander-v2. Higher assist tends to result in better training rewards (right plot), but when assist is too large (e.g. ≥ 1), the agent struggles to learn good policies. If assist is too high, despite good rewards in training, the policy performs quite poorly in the real final environment (left plot), since it is too reliant on high assist. It appears that a small, constant amount of assist $\alpha = .0001$ can actually modestly improve policies over policies learned without any assist whatsoever.

4.2 FetchReach-v1

For the FetchReach environment, we used the same search space as in part 1, but increase the maximum assist level to 10, since the assist is intrinsically scaled by .05 in the fetch-reach environment, as previously mentioned. For our evaluations, we use the sparse reward mechanism.

As shown in figure 5 in the sparse reward setting the importance of assist goes up tremendously - assist can make the difference between the algorithm being able to learn to successfully make reaches within 60 training iterations vs having it fail completely.

We also see a compelling but sensible assist pattern: the amount of assist begins small, scales up when the agent starts learning, then slowly drops down so that the agent is able to autonomously complete the task, almost perfectly.

We also evaluated the assist paradigm using an off policy algorithm, Deep Deterministic Policy Gradients (DDPG) [20]. We implement DDPG using Ray's [17] RLLib and keep all hyperparameters at the default settings. As shown in figure 6, using assisted trajectories allowed DDPG to successfully learn the FetchReach-v1 task.

5 Discussion

We have shown that assist is beneficial to agent's learning, and that policies learned with assist on train faster than policies without assist on.

While we claim these policies are more efficient, it is worth noting that our method does require training 16 policies in parallel. While this effectively increases the number of samples that need to be collected by a factor of 16, it does not affect the training time, since the samples can be collected in parallel. However, for policies training in the real world (e.g. on a robotic arm), it would be necessary to have parallel implementations of the task and algorithm going at once, which could be expensive.

A key area for improvement in our method would be combining our method with results from recent works like Hindsight Experience Replay (HER) [21], which produces stronger results than DDPG on the fetch environment. Given that our method allows DDPG to learn in an environment with sparse rewards without HER, we expect that adding HER will further accelerate training, but baseline HER results would be improved with a dynamics curriculum, since the true goal state would be identified

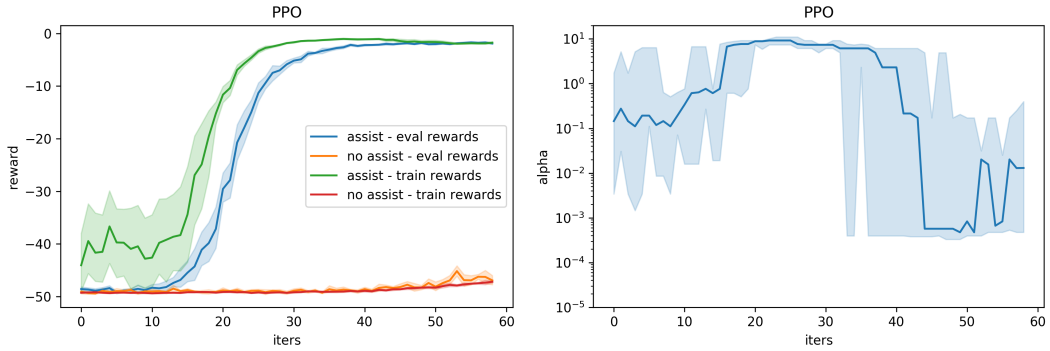

Figure 5: Rewards during evaluation of PPO with and without assist in the FetchReach-v1 environment, and assist schedules for PPO. In a sparse reward environment like FetchReach-v1, assist enables PPO to learn with high speed. As expected, with assist, the reward on the training data is slightly higher than the actual evaluated reward, but then assist is reduced by the scheduler as the agent is able to complete the task.

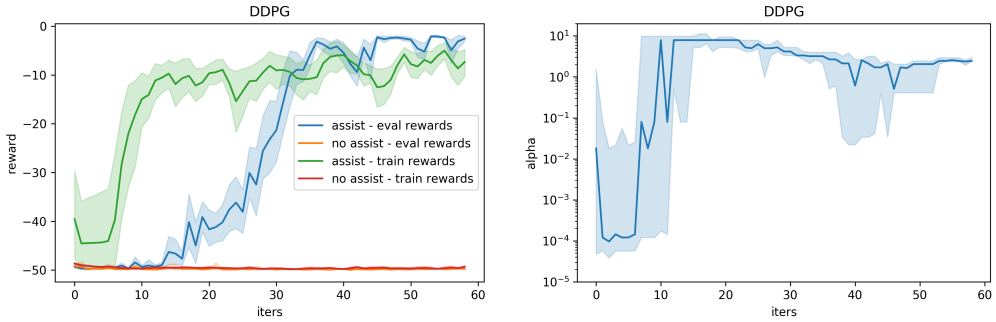

Figure 6: Evaluation of DDPG with and without assist in the FetchReach-v1 environment, and associated assist schedules. We posit that because DDPG has a large amount of low assist trajectories in a replay buffer, it doesn't have to reduce the amount of assist later in training as much as PPO.

more quickly than in HER. Another area for improvement is in combining our results with learning from human demonstrations [22] to hopefully even further improve results and accelerate training.

While we have shown success for these initial simple tasks, it remains to be seen if assist can scale to more difficult tasks. For example, the other tasks in the Fetch robot environment like FetchPush and tasks with multiple goals like FetchPickandPlace would require further engineering of the assist function, since you are not just trying to reach a simple goal. Nevertheless, using assist early in training to complete small subtasks could help improve training.

It would also be compelling to examine if basic forms of assist could be helpful for teaching agents to play video games. For example, in Montezuma's Revenge, which has been shown to be a challenging task for RL agents to solve without exploration models [23]. Nudging the agent towards the reward state early in training could help force it to explore meaningful states earlier in training.

Finally, it would be fascinating to implement assist in the real world via actual forces, e.g. having a robotic arm drawn to specific objects with an electromagnet at the start of training before slowly turning down the electromagnetic force.

6 Conclusions

We have presented assist via dynamics curriculums, which presents a practical way to speed up and improve training of reinforcement learning algorithms. We showed improvements on algorithm convergence time and performance on two domains (video games and robotics). We demonstrated a

recipe for automatically modulating assist levels, and showed that population based approaches can help improve model performance both with and without assist tuning when using policy gradients. Finally, we showed that our assist approach is functional across both on-policy and off-policy methods, and that agents can learn meaningful policies even with small levels of assist on.

References

[1] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[2] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[3] Vikash Gilja et al. "A high-performance neural prosthesis enabled by control algorithm design". In: *Nature Neuroscience* 15.12 (2012), pp. 1752–1757. DOI: 10.1038/nn.3265. URL: https://doi.org/10.1038/nn.3265.

[4] Daniel B. Silversmith et al. "Plug-and-play control of a brain–computer interface through neural map stabilization". In: *Nature Biotechnology* (2020). DOI: 10.1038/s41587-020-0662-5. URL: https://doi.org/10.1038/s41587-020-0662-5.

[5] Yoshua Bengio et al. "Curriculum Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 41–48. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: https://doi.org/10.1145/1553374.1553380.

[6] Jeffery Elman. "Learning and development in neural networks, the importance of starting small". In: *Cognition* (1993).

[7] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.1556.

[8] Alex Graves et al. "Automated Curriculum Learning for Neural Networks". In: *CoRR* abs/1704.03003 (2017). arXiv: 1704.03003. URL: http://arxiv.org/abs/1704.03003.

[9] Carlos Florensa et al. "Reverse Curriculum Generation for Reinforcement Learning". In: *CoRR* abs/1707.05300 (2017). arXiv: 1707.05300. URL: http://arxiv.org/abs/1707.05300.

[10] Carlos Florensa et al. *Automatic Goal Generation for Reinforcement Learning Agents*. 2018. arXiv: 1705.06366 [cs.LG].

[11] Wojciech Marian Czarnecki et al. *Mix Match - Agent Curricula for Reinforcement Learning*. 2018. arXiv: 1806.01780 [cs.LG].

[12] Max Jaderberg et al. *Population Based Training of Neural Networks*. 2017. arXiv: 1711.09846 [cs.LG].

[13] Daniel Ho et al. "Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules". In: *CoRR* abs/1905.05393 (2019). arXiv: 1905.05393. URL: http://arxiv.org/abs/1905.05393.

[14] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[15] John Schulman et al. "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

[16] Max Jaderberg et al. "Population Based Training of Neural Networks". In: *CoRR* abs/1711.09846 (2017). arXiv: 1711.09846. URL: http://arxiv.org/abs/1711.09846.

[17] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications". In: *CoRR* abs/1712.05889 (2017). arXiv: 1712.05889. URL: http://arxiv.org/abs/1712.05889.

[18] Greg Brockman et al. "OpenAI Gym". In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[19] Sid Reddy, Anca Dragan, and Sergey Levine. "Where Do You Think You're Going?: Inferring Beliefs about Dynamics from Behavior". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018, pp. 1454–1465. URL: https://proceedings.neurips.cc/paper/2018/file/6f2268bd1d3d3ebaabb04d6b5d099425-Paper.pdf.

[20] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1509.02971.

[21] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *CoRR* abs/1707.01495 (2017). arXiv: 1707.01495. URL: http://arxiv.org/abs/1707.01495.

[22] Ashvin Nair et al. “Overcoming Exploration in Reinforcement Learning with Demonstrations”. In: *CoRR* abs/1709.10089 (2017). arXiv: 1709.10089. URL: http://arxiv.org/abs/1709.10089.

[23] Adrien Ecoffet et al. “Go-Explore: a New Approach for Hard-Exploration Problems”. In: *CoRR* abs/1901.10995 (2019). arXiv: 1901.10995. URL: http://arxiv.org/abs/1901.10995.