Massively Scaling Explicit Policy-conditioned Value Functions

Nico Bohlinger
Department of Computer Science
Technical University of Darmstadt, Germany
nico.bohlinger@tu-darmstadt.de

Jan Peters
German Research Center for AI (DFKI)
Hessian.AI
Centre for Cognitive Science
Department of Computer Science
Technical University of Darmstadt, Germany
jan.peters@tu-darmstadt.de

Abstract

We introduce a scaling strategy for Explicit Policy-Conditioned Value Functions (EPVFs) that significantly improves performance on challenging continuous-control tasks. EPVFs learn a value function $V(\theta)$ that is explicitly conditioned on the policy parameters, enabling direct gradient-based updates to the parameters of any policy. However, EPVFs at scale struggle with unrestricted parameter growth and efficient exploration in the policy parameter space. To address these issues, we utilize massive parallelization with GPU-based simulators, big batch sizes, weight clipping and scaled peturbations. Our results show that EPVFs can be scaled to solve complex tasks, such as a custom Ant environment, and can compete with state-of-the-art Deep Reinforcement Learning (DRL) baselines like Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). We further explore action-based policy parameter representations from previous work and specialized neural network architectures to efficiently handle weight-space features, which have not been used in the context of DRL before.

Keywords: Deep Reinforcement Learning, Scaling Laws, Value Functions

Acknowledgements

This project was funded by National Science Centre, Poland under the OPUS call in the Weave program UMO-2021/43/I/ST6/02711, and by the German Science Foundation (DFG) under grant number PE 2315/17-1.

1 Introduction

The remarkable success of deep learning in recent years is closely linked to the concept of *scaling*. In particular, the scaling of neural networks, datasets, and computational resources has led to significant advances in various domains such as computer vision and natural language processing. Hardware improvements for GPUs and TPUs and parallelization techniques like data and model parallelism have enabled the training of large-scale models on massive datasets. The relationship between scaling and improved performance has been formalized into the concept of *scaling laws* [1, 2].

While the benefits of scaling are well-established in supervised learning, its application to Deep Reinforcement Learning (DRL) is less understood. Naively increasing the size of policy and value function networks and using bigger batch sizes often leads to diminishing returns or even performance degradation [3]. However, recent work has shown that carefully designing the neural network architecture by incorporating normalization layers, like LayerNorm, BatchNorm or WeightNorm, and residual connections can help to scale up DRL algorithms and improve their performance with bigger networks [4, 5, 6, 7]. On-policy algorithms like Proximal Policy Optimization (PPO) have been shown to greatly benefit from bigger batch sizes, which can be collected efficiently using up to thousands of parallel environments [8].

In this work, we investigate the scaling capabilities of different variations of Explicit Policy-conditioned Value Functions (EPVFs) with the help of massively parallel environments. While EPVFs have previously struggled on complex tasks, we show the importance of scaling and weight regularization for the training of EPVFs and compare different neural network architectures and training setups on a MuJoCo Ant and Cartpole environment.

2 Explicit Policy-conditioned Value Functions

In Reinforcement Learning (RL), we consider a Markov Decision Process (MDP) defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition dynamics, R is the reward function, γ is the discount factor, and ρ_0 is the initial state distribution. The goal of an RL agent is to learn a policy $\pi_{\boldsymbol{\theta}}(a|s)$ that is parameterized by $\boldsymbol{\theta}$. Rolling out the policy in the environment generates a trajectory $\tau = (s_0, a_0, r_0, \ldots)$, where s_0 is sampled from ρ_0 and $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$. The return R_t is defined as the sum of discounted rewards $R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$, where T is the time horizon. The policy is trained to maximize the expected return $J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}, s_0\sim\rho_0}[R_0]$. The state-value function for the policy is defined as $V^{\pi_{\boldsymbol{\theta}}}(s) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_t|s_t = s]$, with which we can re-formulate the policies objective as

$$J(\pi_{\boldsymbol{\theta}}) = \int_{\mathcal{S}} \rho_0(s) V^{\pi_{\boldsymbol{\theta}}}(s)\, ds. \tag{1}$$

Many RL algorithms use the action-value function $Q^{\pi_{\boldsymbol{\theta}}}(s,a) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R_t|s_t = s, a_t = a]$ to decompose the state-value function in the policy objective into $V^{\pi_{\boldsymbol{\theta}}}(s) = \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s,a)\, da$ to get the gradient w.r.t. $\boldsymbol{\theta}$ to optimize the policy. EPVFs take a different approach by learning a value function that is explicitly conditioned on policy parameters or some other differentiable representation of the policy $V(\boldsymbol{\theta})$, to directly optimize the gradient of the policy objective [9, 10, 11]:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \int_{\mathcal{S}} \rho_0(s) \nabla_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta})\, ds = \mathbb{E}_{s\sim\rho_0}[\nabla_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta})] = \nabla_{\boldsymbol{\theta}} V(\boldsymbol{\theta}). \tag{2}$$

The value function $V(\boldsymbol{\theta})$ is learned using a replay buffer of policy parameters and returns, which can be collected by any policy, and updated using stochastic gradient descent. Because the value function predicts the performance of a policy from start to finish and we consider the episodic setting, the target is the undiscounted return. The policy is then updated by following the gradient of the value function with respect to the policy parameters. The resulting algorithm is presented in Algorithm 1.

Algorithm 1 Actor-Critic with Explicit Policy-conditioned Value Function

Input: Initial policy parameters $\boldsymbol{\theta}$, initial value function parameters $\boldsymbol{\phi}$, replay buffer D
for I steps **do**
  Choose M policy parameters or representations $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$
  Rollout and compute undiscounted return R_m for each $\boldsymbol{\theta}_m$
  Store $\{(R_1, \boldsymbol{\theta}_1), \ldots, (R_M, \boldsymbol{\theta}_M)\}$ in D
  for K steps **do**
    Sample batch $B = \{(R, \boldsymbol{\theta})_1, \ldots, (R, \boldsymbol{\theta})_N\}$ from D
    Update $\boldsymbol{\phi}$ with gradient descent: $\nabla_{\boldsymbol{\phi}} \frac{1}{N} \sum_{n=1}^{N} [R_n - V_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n)]^2$
  end for
  for L steps **do**
    Update $\boldsymbol{\theta}$ with gradient descent: $\nabla_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} -V_{\boldsymbol{\phi}}(\boldsymbol{\theta})$
  end for
end for

EPVFs have the key advantage to learn a value function that can potentially reason about all possible policies instead of just the current policy and directly optimize the policy through the value function network. This enables completely off-policy or offline learning with any kind of policy data. Data can be collected by policies that might be optimized for different objectives as long as the associated return for rolling out the policy is known. This can be especially useful in multi-task settings or settings where exploration with different strategies is needed.

When using the concatenation of the raw policy parameters as the input to the value function, the size of the value function network can quickly blow up with the number of policy parameters. This can be mitigated by using another representation of the policy, like the concatenation of actions given a set of probing states [9, 11]. Furthermore, specialized neural network architectures that are designed to handle weight-space features efficiently, by utilizing the symmetry properties of neural network weights [12, 13, 14], are another interesting direction to prevent the network from blowing up in size. These architectures have not been used in the context of RL or even continual online learning before. We investigate both alternatives and more in the following experiments.

3 Experiments

First, we evaluate the performance of EPVFs on the Gymnasium Cartpole environment [15]. From previous work, we know that EPVFs can fully solve simple tasks like Cartpole without additional scaling, i.e., using a small batch size of 16, a replay buffer of size $1e5$, a two layer Multilayer Perceptron (MLP) with 64 neurons for the deterministic policy, and only a single environment [10]. For choosing the policy parameters during rollout, we follow Faccio et al. [10] and simply use the current best policy parameters and perturb them with Gaussian noise $\mathcal{N}(\mu = 0, \sigma = 1.0)$. We compare the single environment setting with using up to 16 parallel environments, where the policy parameters for every environment are perturbed with a different sample of Gaussian noise, i.e. we use as many different policies as environments. Figure 1 shows that scaling the number of environments, and therefore the number of different policies, improves convergence speed, while the perturbed policies in all settings eventually reach the same maximum return of 500. As the Cartpole task is not particularly challenging, we observe diminishing returns with more than 8 environments.

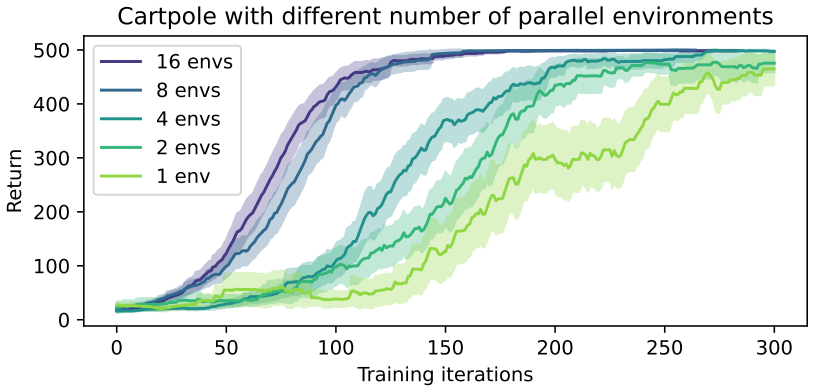

Figure 1: Performance on Cartpole with different numbers of parallel environments. The return is the average undiscounted return achieved with the perturbed policy parameters used during data collection.

Next, we evaluate the performance of EPVFs when scaling to massively parallel environments on a custom Ant environment. For the physics simulation, we us MJX, which is a highly parallelizable GPU-based version of MuJoCo [16] based on JAX [17]. Our Ant environment is a continuous control task with a 34-dimensional state space and a 8-dimensional action space. In this task, the Ant's objective is to walk forward at a target velocity of 2 m/s. The time horizon is set to 1000 steps and the reward is calculated as $r = \exp(-|v_{xy} - c_{xy}|^2/0.25)$, where v_{xy} is the linear velocity of the Ant and c_{xy} is the target velocity $(2, 0)$. This results in a maximum possible return of 1000. We setup the learning environment and algorithm in the DRL framework RL-X [18]. We can jit-compile the full training loop, enabling up to 4096 parallel environments on a single RTX 3090 GPU. This setup allows for extremely fast data collection and throughput of up to 3 million environment steps per second.

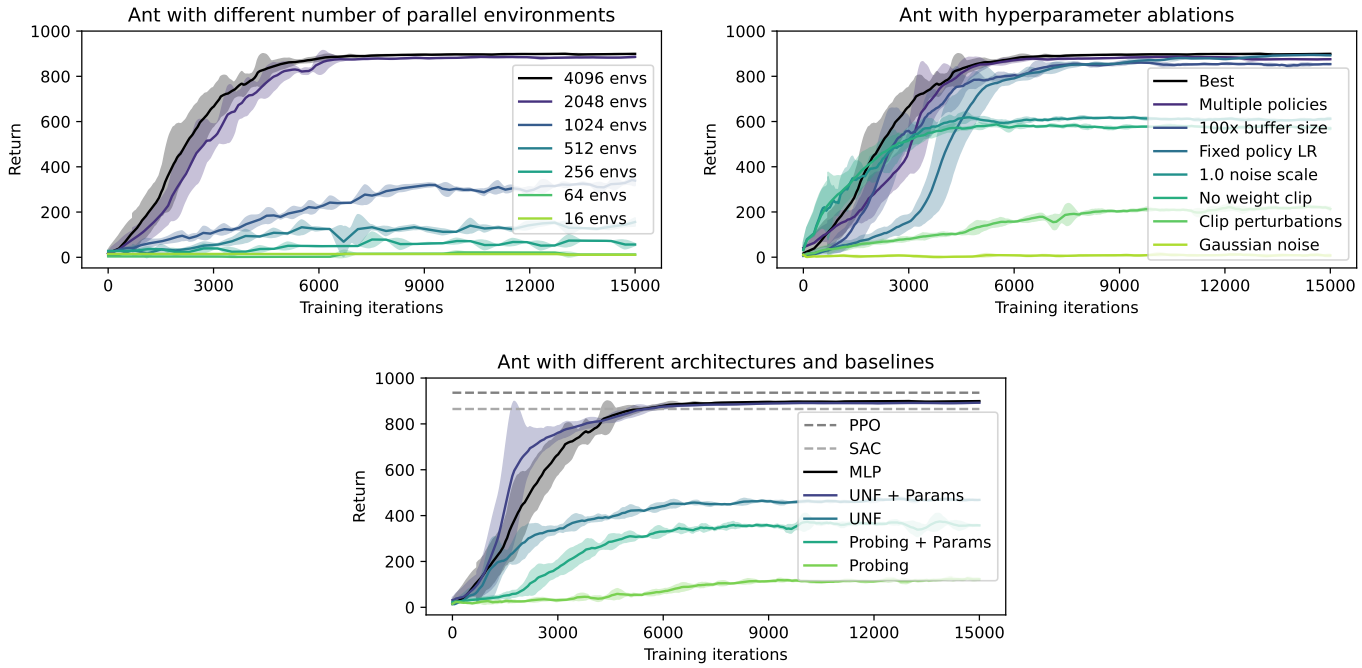

Figure 2: *Top left* – Performance on Ant with different numbers of parallel environments, which equal the batch size. *Top right* – Ablation on the different algorithmic changes introduced to scale EPVFs. Every ablation uses all the same parameters as the best setup but with one change. Multiple policies: Update a set of 4096 differently initialized policy parameters instead of just one. 100x buffer size: Increase the replay buffer size to 409600 to sample older data as well. Fixed policy LR: Use a fixed learning rate of $1e-5$ for the policy instead of the learning rate schedule. 1.0 noise scale: Use a uniform noise scale of 1.0 for perturbing the policy parameters instead of 0.3. No weight clip: Do not clip the policy parameters. Clip perturbations: Directly clip the perturbations of the policy parameters to $(-0.3, 0.3)$ instead of only after the gradient steps. Gaussian noise: Use $\mathcal{N}(\mu = 0, \sigma = 1.0)$ instead of uniform noise for perturbing the policy parameters. *Bottom* – Performance of the action-based policy parameter representations (Probing) and specialized weight-space architectures (UNF). Additionally, the performance of the PPO and SAC baselines is shown as dashed lines.

To successfully train EPVFs on the Ant environment, the training setup is significantly scaled up. We increase the number of environments to 4096, the batch size to 4096 and also the replay buffer size to 4096. As the replay buffer is now just as big as the batch size, the value function is trained on only fresh data and the big batch size ensures good gradient estimates through averaging on diverse data. Furthermore, we perturb the policy parameters with uniform noise that is $(-0.3\omega, 0.3\omega)$ for each parameter ω instead of Gaussian noise. This change allows the added noise to be automatically scaled to the magnitude of every policy parameter. Additionally, we introduce weight clipping for the policy parameters to be in the range $(-0.1, 0.1)$, which helps to stabilize the training and prevent the policy parameters from constantly growing in magnitude. Alternatively, weight regularization with strong weight decay can also be used but the weight decay coefficient is harder to tune. Finally we add an exponential learning rate schedule for the policy from $1e-3$ to $1e-7$ over the course of training. The top right of Figure 2 shows the ablations on the mentioned algorithmic changes.

The top left of Figure 2 highlights the scaling capabilities of EPVFs on the Ant environment, where the number of environments equals the batch size. The results show that increasing the batch size directly improves the policy performance. An environment and batch size of at least 2048 is necessary to achieve strong performance and to solve the task, while setups with less environments struggle to reach even a quarter of the maximum return. EPVFs thrive in settings with many environments and large batch sizes, as the value function trained for the policy parameter space is inherently more difficult to learn but can give good gradient estimates to the policy when the policy parameter space is well explored and well restricted with weight clipping.

Finally, we investigate action-based policy parameter representations and specialized neural network architectures for the value function and also compare the performance of EPVFs to PPO and Soft Actor-Critic (SAC) [19]. For the action-based policy parameter representation, we follow prior work [9, 11] and use the concatenation of actions given a set of 200 learnable probing states as the input to the value function. Furthermore, we test a variation where the resulting action representation is concatenated with the raw policy parameters to see if both representations can be combined. Figure 2 shows that the action-based representation is not reaching the same performance as just using the raw policy parameters. The combination of both improves upon only using the actions but still does not reach good performance.

For the specialized weight-space neural network architecture, we use the recently proposed Universal Neural Functional (UNF) [14] for the value function. Figure 2 shows that the features from the UNF architecture in concatenation with the raw policy parameters are able to reach the same performance as the default MLP architecture, while even learning slightly faster. When using UNF features alone, the learned policy is not able to reach the same end performance.

4 Conclusion

We have shown that EPVFs can be scaled to solve complex continuous control tasks and compete with state-of-the-art DRL baselines with the help of massively scaling up the number of parallel environments and the batch size. Key to stability is the use of weight clipping, to restrict the policy parameter space and prevent the parameters from constantly growing, and the use of uniform noise scaled to the magnitude of the parameters for exploring the policy parameter space efficiently. Further investigation on weight-space features from specialized architectures like UNFs might enable even better scaling capabilities for EPVFs in the future.

References

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *preprint arXiv:2001.08361*, 2020.

[2] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. In *36th Conference on Neural Information Processing Systems*, pages 30016–30030, 2022.

[3] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International conference on learning representations*, 2021.

[4] A. Kumar, R. Agarwal, X. Geng, G. Tucker, and S. Levine. Offline q-learning on diverse multi-task data both scales and generalizes. In *The Eleventh International Conference on Learning Representations*, 2023.

[5] A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024.

[6] M. Nauman, M. Ostaszewski, K. Jankowski, P. Miłoś, and M. Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. In *38th Conference on Neural Information Processing Systems*, 2024.

[7] H. Lee, D. Hwang, D. Kim, H. Kim, J. J. Tai, K. Subramanian, P. R. Wurman, J. Choo, P. Stone, and T. Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. *preprint arXiv:2410.09754*, 2024.

[8] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[9] J. Harb, T. Schaul, D. Precup, and P.-L. Bacon. Policy evaluation networks. *preprint arXiv:2002.11833*, 2020.

[10] F. Faccio, L. Kirsch, and J. Schmidhuber. Parameter-based value functions. In *International Conference on Learning Representations*, 2021.

[11] F. Faccio, A. Ramesh, V. Herrmann, J. Harb, and J. Schmidhuber. General policy evaluation and improvement by learning to identify few but crucial states. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*.

[12] A. Navon, A. Shamsian, I. Achituve, E. Fetaya, G. Chechik, and H. Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pages 25790–25816. PMLR, 2023.

[13] A. Zhou, K. Yang, K. Burns, A. Cardace, Y. Jiang, S. Sokota, J. Z. Kolter, and C. Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36, 2024.

[14] A. Zhou, C. Finn, and J. Harrison. Universal neural functionals. In *38th Conference on Neural Information Processing Systems*, 2024.

[15] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *preprint arXiv:2407.17032*, 2024.

[16] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[17] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[18] N. Bohlinger and K. Dorer. Rl-x: A deep reinforcement learning library (not only) for robocup. In *Robot World Cup*, pages 228–239. Springer, 2023.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.