DNA: Proximal Policy Optimization with a Dual Network Architecture

Anonymous Author(s)
Affiliation
Address
email

Abstract

This paper explores the problem of simultaneously learning a value function and policy in deep actor-critic reinforcement learning models. We find that the common practice of learning these functions jointly is sub-optimal, due to an order-of-magnitude difference in noise levels between these two tasks. Instead, we show that learning these tasks independently, but with a constrained distillation phase, significantly improves performance. Furthermore, we find that the policy gradient noise levels can be decreased by using a lower *variance* return estimate. Whereas, the value learning noise level decreases with a lower *bias* estimate. Together these insights inform an extension to Proximal Policy Optimization we call *Dual Network Architecture* (DNA), which significantly outperforms its predecessor. DNA also exceeds the performance of the popular Rainbow DQN algorithm on four of the five environments tested, even under more difficult stochastic control settings.

1 Introduction

Combining deep neural networks with reinforcement learning has produced impressive results on challenging problems, such as playing Chess [26], Atari games [21], and robotics tasks [25]. However, results have bifurcated between two competing approaches: Q-learning-based approaches that require learning an (action conditioned) value estimate and actor-critic policy gradient (AC-PG, or just PG)[1] approaches that learn both a policy and value estimate. PG offers many theoretical advantages over Q-learning, such as natural support for continuous control problems and the ability to learn stochastic policies. However, until recently, PG approaches [20, 25, 9], while strong on continuous control problems, have under-performed Q-learning approaches on complex vision-based problems [21, 14, 3], restricting the use of PG in this domain.

This paper aims to close the gap between PG and Q-learning methods by showing that the common practice of jointly learning value and policy with a shared representation negatively affects the algorithm's performance.[2] We also demonstrate an order-of-magnitude difference in noise levels between these two tasks and argue that this makes these two tasks poorly aligned.

In light of this result, we introduce a new algorithm, based on Proximal Policy Optimization (PPO) [25], called Dual Network Architecture (DNA). We test DNA empirically on a subset of the Arcade Learning Environment (ALE) [5] called Atari-5.[3], Our results show a strong increase in performance compared to both PPO and another dual network model Phasic Policy Gradient (PPG) [9]. Our model

[1] For simplicity, we refer to actor-critic policy gradient as policy gradient, even though some early policy gradient approaches, such as REINFORCE [29], do not learn a value function.

[2] PPO [25], A3C [20], and Muslie [13] all use a joint network when training on vision-based discrete action environments. PPG [9] uses a dual network with an auxiliary task, which we discuss in more detail in Section 2

[3] This is a parallel submission. An anonymous copy is provided in the supplementary materials.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

also outperforms Rainbow DQN [14] on four of the five games tested, even under more challenging environmental settings.

We summarize our contributions. First, we provide empirical results showing an order-of-magnitude difference in the noise scale between policy gradient and value learning. Second, we give evidence for the benefit of using low *bias* value estimates for value learning, but low *variance* estimates for advantage estimates. Finally, we introduce and justify our dual network constrained distillation algorithm DNA with empirical results on ALE.

2 Preliminaries and Related Work

Multi-task learning Multi-task learning is the process of learning a shared representation that can be used to solve multiple tasks in the hope that *constructive interference* between the tasks will lead to better performance [22]. However, this is not always the case. When tasks are unaligned *destructive interference*, can instead reduce the performance of the model [30]. We argue that large differences between the noise scale in value learning and policy gradient make these tasks poorly unaligned.

TD(λ) Return Estimation Policy gradient algorithms often make use of a value estimate as a baseline [20], as well as for estimating the value of truncated trajectories [7]. To better facilitate control over the noise levels for policy and value learning, our work makes use of two different return estimations, both using TD(λ) [27]. Given value estimates $V(\cdot)$ from the value network we define the n-step value estimate for some state s_t taken at time t, and their exponentially weighted sum as

$$\text{NSTEP}^{(\gamma,k)}(s_t) := \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}), \tag{1}$$

$$\text{TD}^{(\gamma,\lambda)}(s_t) := (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \text{NSTEP}^{(\gamma,k)}(s_t). \tag{2}$$

For values of λ close to 1, more weight is assigned to longer n-step return estimates, and less to shorter ones. There has been a long-standing belief that shorter n-step returns generate more biased estimates, whereas longer n-step estimates, due to summing over many stochastic rewards, have higher variance.[4] For a more thorough discussion on this topic see [16].

Noise Scale. Noise scale is a measure of 'signal to noise' in stochastic gradient descent (SGD). While SGD provides unbiased gradient estimates, these estimates are typically very noisy. Each sampled gradient estimate $\hat{G}$ can be thought of as a sum of the true gradient $\vec{G}$ and some noise vector $\vec{\sigma}$. A method for estimating the ratio of the magnitude of this implied noise vector to the magnitude of the true gradient was proposed by [19] who show that their efficient-to-calculate *simple noise scale* is a good match for the noise scale. They also show that the noise scale provides useful information about the choice of mini-batch size to use when estimating gradients.

Phasic Policy Gradient. Most similar to our work is Phasic Policy Gradient (PPG) [9]. Like DNA, PPG has three distinct phases during training and uses two independent networks. However, there are several important distinctions in our work. First, we forgo the large replay buffer, relying instead on learning entirely from recent experience, significantly reducing the memory requirements of our algorithm. Second, we make use of a distillation phase rather than an auxiliary phase.[5] Finally, our work reduces gradient noise by using two different return estimators calibrated for the properties of each task. We assess the impact of these differences on the performance of the agent in Section 6.

[4] While there are cases (i.e. where rewards have negative covariance) where this is not the case, empirical experiments confirm the intuition that longer n-step estimates generally have higher variance and lower bias.

[5] PPG refer to their third phase as a distillation phase, however, because the update trains both the policy and value networks on value estimates generated from rollouts, this is better thought of as an auxiliary task. We discuss this in more detail in Section 4.3

3 The Noise Properties of Value Learning and Policy Gradient

Here we examine the noise properties of the loss functions for policy gradient and value learning and show that policy gradient has a much higher noise level than value learning. This implies different challenges in the optimization problem. Specifically, that policy should be learned with a larger mini-batch size than value learning and that return estimates should be adapted appropriately.

3.1 Motivating Example

To motivate our investigation into noise levels, we consider the task of learning two independent functions $F_1(x) := \sin(5x) + \mathrm{N}(0, \sigma_1^2)$ and $F_2(x) := \cos(5x) + \mathrm{N}(0, \sigma_2^2)$, where $\mathrm{N}(\mu, \sigma^2)$ is Gaussian noise with mean μ and standard deviation σ, over the domain $[-\pi, \pi]$. We fix $\sigma_2 = 1$, and vary σ_1 on a log scale from 0.1 to 100. We trained two 3-layer multi-layer perceptions (MLPs) on this problem.[6] The first was a joint model, using a first hidden layer of 1024 units and a second hidden layer of 2048 units, followed by two output heads. Our second model was a dual network consisting of two independent MLPs with 1024 units on each hidden layer.[7] All models used ReLU activations in between linear layers, and error was measured as mean-squared-error (MSE) between the predicted value and the *noise free* true value. The results are presented in Figure 1. At low noise levels, the tasks did not interfere, but as the noise of the first task increased, performance on the second task eventually degraded for the joint network, but not the dual network. While simple, this experiment demonstrates the impact one noisy task can have on another when learned jointly by a deep neural network.

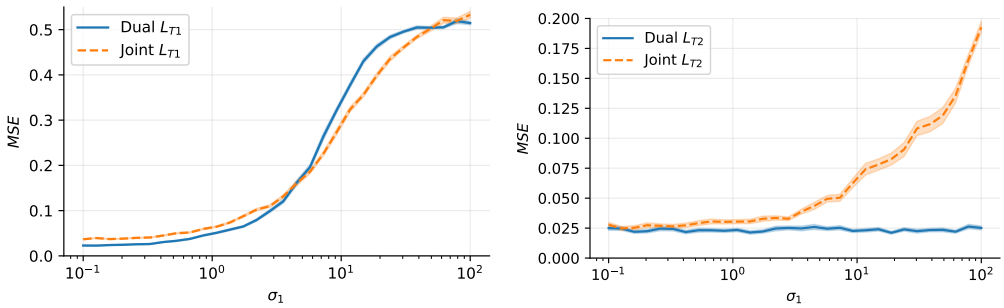

Figure 1: A depiction of the problem of destructive interference on a toy problem. L_{T1} and L_{T2} refer to the mean-squared-error on the first and second task respectively.Shading indicates standard error over 100 seeds.

3.2 Noise Scale in Reinforcement Learning

To assess the differences in noise levels in our experiments, we measure the *gradient noise scale* of the policy gradient and value loss gradient. We did this by learning an estimate of the noise scale, developed by [19] called *simple noise scale* defined as

$$\mathcal{B}_{\text{simple}} := \frac{\text{tr}(\Sigma)}{|\vec{G}|^2}, \tag{3}$$

where Σ is the gradient covariant matrix, and $\vec{G}$ is the true gradient if the entire batch was used. For convenience we also use the notation $\sigma := \sqrt{\mathcal{B}_{\text{simple}}}$, which can interpreted as the ratio of the length of the implied noise vector to the length of the true gradient vector. That is, a kind of noise-to-signal ratio. As [19] have shown, an unbiased estimate of $\mathcal{B}_{\text{simple}}$ can be found efficiently by generating gradient estimates $\vec{G}_{B_{\text{small}}}$ using a small mini-batch of size B_{small} as well as the gradient $\vec{G}_{B_{\text{big}}}$ using a large mini-batch of size B_{big}, then calculating

[6] It is worth noting that we did not find the same result when using shallow networks. That is, dual and joint models performed similarly when using a 2-layer MLP.

[7] This number of hidden units was chosen so that both models had the same number of parameters.

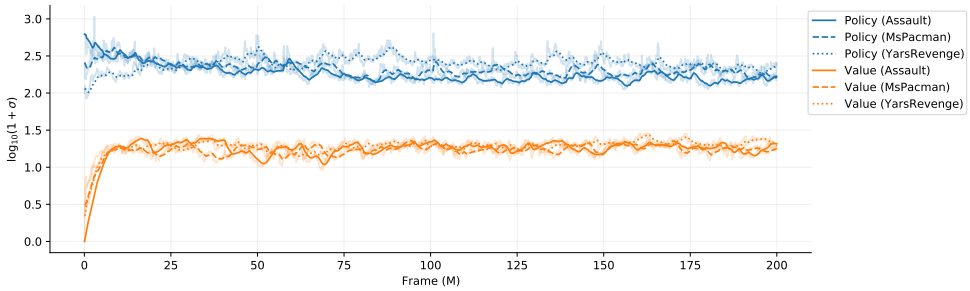

Figure 2: Noise scale for the two learning objectives.For clarity all values are further smoothed using an exponential rolling average (with non-smoothed values presented in a lighter shade). Smoothing may introduce bias, as while $|\mathcal{G}|^2$ and $\mathcal{S}$ are unbiased estimators, their ratio may not be.

$$|\mathcal{G}|^2 := \frac{1}{B_{\text{big}} - B_{\text{small}}}(B_{\text{big}}|\vec{G}_{B_{\text{big}}}|^2 - B_{\text{small}}|\vec{G}_{B_{\text{small}}}|^2) \tag{4}$$

$$\mathcal{S} := \frac{1}{1/B_{\text{small}} - 1/B_{\text{big}}}(|\vec{G}_{B_{\text{small}}}|^2 - |\vec{G}_{B_{\text{big}}}|^2), \tag{5}$$

where $\mathbb{E}[|\mathcal{G}|^2] = |\vec{G}|^2$ and $\mathbb{E}[\mathcal{S}] = \text{tr}(\Sigma)$.

To evaluate the noise scale of value learning and policy gradient we trained our dual network model (described fully in section 4) on the three Atari games from the Atari-3 validation set.[8] Like [19], we found it necessary to smooth out noise by maintaining an exponential moving average of $|\vec{G}_{B_{\text{big}}}|^2$. We used $B_{\text{small}} = 16$ and $B_{\text{big}} = 16,384$ in all our experiments.

The results are presented in Figure 2. We found that policy noise (σ_π) reduced by about half during training and that value noise (σ_v) remained constant. Both noise levels varied very little between the three environments. However, a large variation was observed between the value loss and policy gradient, consistent between environments. The noise levels, measured at the end of training and averaged over all three environments, were found to be $\sigma_\pi = 220.5$ and $\sigma_V = 17.5$, representing a 12.6 times difference.

4 Dual Network Architecture

Based on our experimental results in Section 3 we propose an architecture which takes into account the large difference in noise levels between the two tasks. This architecture consists of three improvements to PPO. First, to reduce negative interference from the noisy policy gradient, policy and value should be learned by independent networks with different hyperparameters (e.g. training epochs and mini-batch size). Second, the variance/bias trade-off in the return estimations should be calibrated to the properties of each task. Finally, a constrained distillation phase should account for any constructive interference between the tasks.

4.1 Independent Networks

Important to DNA is the use of a dual network architecture.[9] Not only does this setup allow for learning of the value and policy without destructive interference, it also enables a specialized set of hyperparameters to be calibrated for the distinct tasks of policy and value learning. Specifically, the calibration of the mini-batch size, which it has been suggested is best set roughly proportional to σ [19]. The policy network outputs a policy π and a value estimate V_π, whereas the value network only outputs a value V_V, as depicted in Figure 3.

[8] See our parallel Atari-5 paper, included in the supplementary material.

[9] By dual network, we mean two independent networks. Not to be confused with a dual-head network (which we refer to as a joint network), or with dualing networks [28].

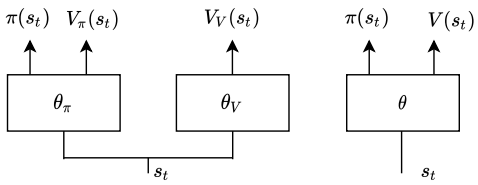

Figure 3: Architecture of DNA (left) compared to a single network setup (right).

4.2 Decoupled Return Estimation

To better facilitate control over the noise levels for policy and value learning, DNA makes use of two different return estimations, both using TD(λ) [27] as follows

$$V_{\text{targ}}(s_t) := \text{TD}^{(\gamma,\lambda_V)}(s_t), \tag{6}$$

$$V_{\text{adv}}(s_t) := \text{TD}^{(\gamma,\lambda_\pi)}(s_t), \tag{7}$$

where λ_V and λ_π are hyperparameters controlling the variance / bias trade-off of each estimate. The V_{targ} estimates are used as targets for training the value function, whereas V_{adv} are used for advantage estimates used for policy gradient estimates given by

$$\hat{A}_t := V_{\text{adv}}(s_t) - V(s_t). \tag{8}$$

This formulation of the advantages is equivalent to the general advantage estimation (GAE) [24], that is, for any $\lambda \in [0..1], \gamma \in [0..1)$ we have,[10]

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \text{TD}^{(\gamma,\lambda)}(s_t) - V(s_t). \tag{9}$$

Because of this, it is common when using PPO to generate value targets, V_{targ}, for the value function by adding the network's value estimates to the (unnormalized) advantages, implicitly settings $\lambda_V = \lambda_\pi$.[11] We hypothesize that due to bootstrapping, estimates used for V_{targ} benefit from being *low bias*, conversely, because estimates used for advantage estimation contribute to the noise of the policy gradient, they would benefit from being *low variance*. This suggests $\lambda_\pi < \lambda_V$, a thought which we test empirically in section 6.2.

4.3 Distillation

Because value learning is a simpler task in terms of noise level, it makes sense to try to transfer some of the knowledge learned by the value network to the policy network. If the value network learns to identify important features in the environment, we would like the policy network to identify those features also. To facilitate this, we employ a constrained distillation update [15]. Distillation between two identical networks has the property that the student's target function since an identical network generated it, is guaranteed to be in the model's solution space. Also, because the distillation targets are deterministic,[12] the process is also very low noise, which we verify in Appendix D.

We contrast this with the auxiliary update of PPG which trains both the value function and the policy on an auxiliary task. This auxiliary task equates to learning $\frac{1}{N}\sum_{i=0}^{N} V_{\text{targ}}^{\pi_{-i}}$ where $V_{\text{targ}}^{\pi_{-i}}$ is a value estimate for the policy from i updates prior. For quickly changing policies, and large N, these updates may cause tension with the value estimates of the current policy.[13]

[10] A proof for this claim is provided in Appendix E

[11] For example, see line 65 of the popular baselines implementation of PPO. https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/runner.py

[12] That is to say, $V_{\text{targ}}(s_t)$ are samples drawn from the random variable $R(s_t)$, where R is a (noisy) return estimator, whereas $V(s_t)$ is a deterministic function that produces an estimate of $\mathbb{E}[R(s_t)]$

[13] A better way to do auxiliary tasks would be to train a separate value network output head, dedicated to the auxiliary task. This way the agent could learn the value of the current policy, along with the moving average value. We believe, however, that the better solution to overfitting the value function to recent data is simply to train less. An idea which we discuss further in Section 7.

We use $V_V(s)$ as the distillation targets, with the input states s being taken from the current rollout. However, unlike policy and value training, distillation state inputs need not be generated on-policy. We explored other distillation targets, covered in Appendix J, but found $V_V(s)$ to be the best of those tried. Distillation is performed, like PPG's auxiliary task, using mean squared error, and under a soft constraint on the policy network's policy, specifically

$$L_t^D(\theta) := \hat{\mathbb{E}}_t\big[(V_\pi(s_t) - V_V(s_t))^2\big] + \beta \cdot \hat{\mathbb{E}}_t\big[\mathrm{KL}(\pi_{\text{old}}(\cdot|s_t), \pi(\cdot|s_t))\big] \tag{10}$$

where β is the policy constraint coefficient, and π_{old} is a copy of the policy before the distillation update. During updates gradients are only propagated through the policy network, and not the value network.

4.4 Training

Like Phasic Policy Gradient (PPG) [9], DNA splits training into three distinct phases, but unlike PPG, rather than using a large replay buffer, we perform all updates on-policy on the current batch of rollout data. Each of the three phases optimizes a single objective, for some number of epochs, using its own optimizer, with a unique set of hyperparameters. The optimization objective for the policy network of DNA is the clipped surrogate object from PPO [25] including the entropy bonus

$$L_t^{\text{CLIP}} := \hat{\mathbb{E}}_t\Big[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) + c_{\text{eb}} \cdot \mathrm{S}[\pi(s_t)]\Big] \tag{11}$$

where S is the entropy in nats, ρ_t is the ratio $\frac{\pi(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ at time t, ϵ is the clipping coefficient, and c_{eb} is the entropy bonus coefficient. For value loss we use the squared-error loss,

$$L_t^{VF} := \hat{\mathbb{E}}_t\big[(V_V(s_t) - V_{\text{targ}}(s_t))^2\big]. \tag{12}$$

In summary, the DNA algorithm separates the tasks of policy learning and value learning into a network designed to handle the high noise of policy learning and a separate network designed to handle the lower noise task of value learning. Knowledge from the value learning network is transferred to the policy network through a separate constrained distillation phase, which allows for constructive interference between the learning tasks while minimizing the destructive. We formalize the algorithm as follows.

Algorithm 1 Proximal Policy Optimization with Dual Network Architecture

```
procedure PPO-DNA

    for t = 1 to N do
        for i = 1 to A do
            Run policy π_t in environment i for T timesteps
        Compute V_targ ← TD^(γ,λ_V)
        Compute Â ← TD^(γ,λ_π) − V_adv
        for i = 1 to E_π do
            Optimize L^CLIP wrt θ_π
        for i = 1 to E_V do
            Optimize L^VF wrt θ_V
        π_old ← π_t
        for i = 1 to E_D do
            Optimize L^D wrt θ_π
```

5 Evaluation

To evaluate the performance of our algorithm, we used the Atari-5 benchmark. Scores on this subset, when properly weighted, correlate well with the median score performance of an algorithm if it had

been run on all 57-games. In all cases, we fit hyperparameters to the 3-game validation set and only used the 5-game test set for final evaluations. To verify the statistical significance of our results, we performed multiple seeded runs.

We opted for the more difficult stochastic ALE settings recommended as best practice by [18]. However, to better understand our results in the context of prior work, we also provide detailed results under the simpler deterministic settings in Appendix F, and additionally provide for reference a single seed evaluation on the full 57-game set in Appendix K. Unless otherwise specified, agents were scored according to their average performance over the previously completed 100-episodes at the end of training.

A coarse hyperparameter sweep found initial hyperparameters for our model on the Atari-3 validation set. Notably, we found the optimal mini-batch size for policy and distillation to be the minimum we tested (256), while the optimal mini-batch size for policy was the largest tested (2048). For optimization, we used Adam [17], over the standard 200 million frames. Full hyperparameter details for our experiments are given in Appendix B.

In order to understand the impact of the hyperparameters introduced by our algorithm, namely E_π, E_V, E_D and λ_π, λ_V, we performed the following experiments on the Atari-3 validation set. We started by setting $E_\pi = E_V = E_D = 2$, then searched over $E_V \in [1, 2, 3, 4]$, and selected the best E_V. We then searched over E_π and E_D while keeping the other two parameters constant. We also performed a similar experiment on the impact of λ_V and λ_π by setting λ_V to 0.95 then sweeping across $\lambda_\pi \in [0.6, 0.8, 0.9, 0.95, 0.975]$, then setting λ_V to the best λ_π and repeating. This process allowed us to verify if the best settings occur at $\lambda_V = \lambda_\pi$ or at $\lambda_V \neq \lambda_\pi$. During our λ tuning, we also recorded noise levels for one of the seeds, the results of which we discuss in Section 6.2.

We evaluated our algorithm DNA, against PPO, and PPG. All models used the 'NatureCNN' encoder from [21]. However, because DNA and PPG use two networks and thus twice the parameters, we doubled the number of channels of the PPO encoder which we refer to as PPO (2x) and which is very similar to the encoder used by [3]. All models have have approximately 3.5 million parameters in total. For fairness we repeated the same sweep on PPO for the number of training epochs (E_{PPO}) and λ used for GAE (λ_{PPO}). To verify that our settings did not inadvertently degrade the performance of PPO we also tested against the settings used by [25], and refer to this as PPO (original). These results are included in the ablation study in Appendix C.

6 Main Experimental Results

In this section we present the results of our main study comparing PPO to DNA and PPG on Atari-5.

6.1 Impact of Training Epochs

We found that using *less* than three policy epochs for DNA dramatically increased the performance of the agent while also decreasing the computation required to train the agent (Figure 4). We also found that DNA was robust to the choice of value and distillation epochs but that overtraining on value ($E_V = 4$) marginally decreased performance. This was not the case for distillation updates. The distillation phase demonstrated some benefit as indicated by the $E_D = 0$ results under-perform the others. We selected $E_\pi = 2, E_V = 1, E_D = 2$ as the optimal settings and used these for the remainder of the experiments. These settings require a total of four updates for the policy network, but only one update for the value network. The training curves also suggest that DNA may benefit from training beyond the standard 200-million frames.

6.2 Return Estimation

We found that the choice of λ_π made a large difference to the performance of our agent. As was hypothesised the optimal settings for λ_π differed to those of λ_V with λ_π preferring a lower variance, higher bias value of 0.8, while the value targets, λ_V preferred a lower bias, higher variance setting of 0.95. Our results are presented in Figure 5. Notable is that the non-homogeneous setting $\lambda_V = 0.95, \lambda_\pi = 0.8$ outperformed both homogeneous choices $\lambda_\pi = \lambda_V = 0.8$ and $\lambda_\pi = \lambda_V = 0.95$. We found, as expected, that noise scale can be reduced by selecting a low value for λ_π. However, against our expectations, setting λ_V lower actually increased, not decreased the

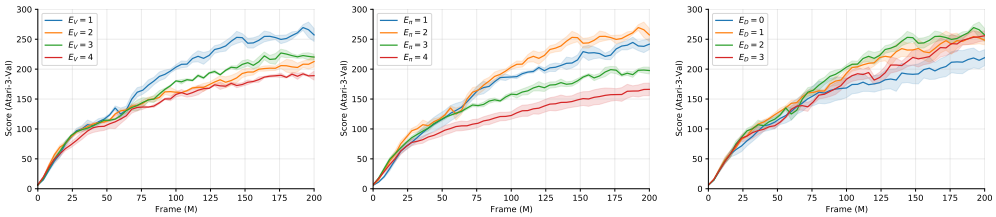

Figure 4: Training curves for DNA with various epochs. The DNA algorithm is most sensitive to the number of policy epochs, and is optimal on our validation set with $E_\pi = 2$. Applying policy updates 3 or 4 times significantly reduces the agents performance.

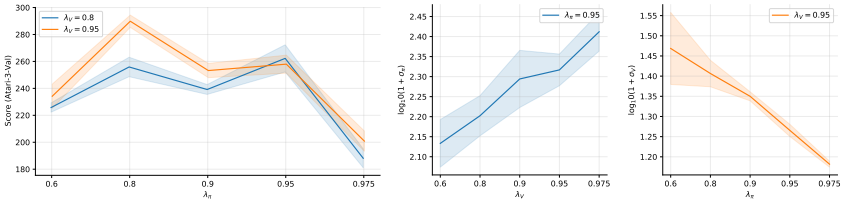

Figure 5: Performance of DNA across a range of λ values for return estimations. Left: The optimal settings for λ_V and λ_π differed. Shading indicates standard error over 5 seeds. Mid/right: σ_π increased with λ_π where as σ_V decreased with λ_V. Shading indicates standard deviation over the environments.

noise level of the value learning task. Once we discovered this we ran addition experiments for a broader choice of λ_V, and include the results in Figure 5.

6.3 Comparision to PPO and PPG

We found DNA to outperform both PPO and PPG on the Atari-5 dataset by a wide margin (Figure 6). We give Atari-5 summary scores, as well as training plots for each individual game. We also include references for Rainbow DQN, although note that these scores were generated under the simpler deterministic settings.

7 Discussion

Reduced Training as an Alternative to Experience Replay In this subsection, we discuss the surprising result that *less* training, both on policy and value updates, typically improves the agent's performance. One explanation for this could be, as [9] have suggested, that in the value case, this might be due to overfitting to recent experience.[14] In terms of policy updates, where the largest performance degradation was observed, this could be due to increased training epochs causing large changes in policy, which may suggest a better constraint than PPO's clipping is required.[15] That being said, limiting the training epochs to one or two provides straightforward solution to the problem while having the additional benefit of decreasing the computational resources required to train an agent. Even in Deep Q-learning, others have found that, beyond some point, increasing the number of times an example is processed has a negative effect unless the learning rate is proportionally reduced [1].

Choice of λ Our results have shown strong evidence for the benefit of selecting $\lambda_\pi < \lambda_V$. As expected, picking *lower* values of λ_π reduces the policy gradient noise. However, counterintuitively, noise on value learning reduces with *higher* choices of λ_V. That is, policy learning prefers low *variance*, but value learning prefers low *bias*. This result may explain why clipping, which introduces

[14] This is plausible, especially with high variance returns

[15] It is worth noting that PPO does clip the entropy bonus, and therefore updates with a non-zero entropy bonus could make arbitrarily large changes to the policy regardless of the choice of ϵ.

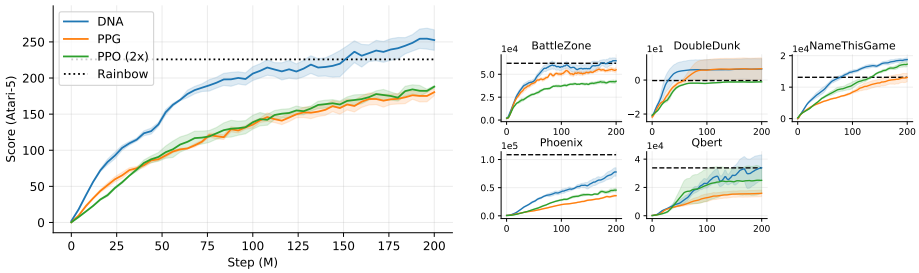

Figure 6: Left: Results on the Atari-5 benchmark. DNA outperforms PPO by a wide margin, and even exceeds the Rainbow algorithm despite the more difficult settings used in our experiment. Right: Individual training curves for each of the five games.

bias, performs poorly on value learning but quite well on policy learning.[16] This result also suggests that an even higher choice of λ_V than the 0.95 we used in our experiments may be appropriate.

Performance of PPG on Atari Our algorithms shares a lot in common with the PPG algorithm, which has shown strong results on the Procgen [8] environment. However, PPG performed quite poorly on the Atari games tested, only marginally outperforming the PPO reference. We suspect that this is due to the auxiliary task involving learning stale values estimates. We verify in the Appendix C that the poor performance was not due to hyperparameter differences.

Performance of PPO on Atari Perhaps the most surprising result of our study was that PPO could perform well on Atari by simply increasing the mini-batch size, and reducing the training epochs down to 1. These settings resulted in an algorithm that was 4-times faster to train than the settings used by [25], and produce much stronger results (Appendix C).

Limitations Our paper's primary focus was improving PG results on discrete action vision-based problems. We do not cover continuous control problems and leave this for future work.

Broader Impact In our work, we have shown that deep actor-critic reinforcement learning models can effectively solve complex vision-based, discrete-action environments, even without the use of large replay buffers. Surprisingly this can be achieved using *less* not more policy and value network updates. Because Q-learning approaches generally produce deterministic policies, algorithms based on them may 'lock in' to a decision based on minimal differences in outcome. This is not always the case with stochastic policies, which are able to randomly split over actions of roughly equal quality. This may lead to fairer outcomes. We do not foresee any direct negative societal impacts from this research.

Future work Our algorithm is already highly competitive with Rainbow DQN, and could likely be extended with orthogonal improvements such as adding recurrence [12], distributional value learning [4], residual encoders [11], better exploration bonuses [6], and making use of a learned model [23].

8 Conclusions

In this paper, we have highlighted noise level as a key difference between policy and value learning. We have introduced an algorithm that accounts for this order-of-magnitude difference by limiting *negative* interference through two independent networks for value and policy but retaining *constructive* interference through a constrained distillation process. Furthermore, we have shown that the variance/bias tradeoff differs for value learning and policy gradient and that return estimation should cater for this. Together these changes result in a novel algorithm, DNA, that outperforms its predecessor PPO, and even surpasses the popular Q-learning approach Rainbow DQN, while under more challenging environmental settings.

[16] By value clipping, we mean the value trust-region suggested by [24], but which has been shown hurt performance by [2].

References

[1] M. Aitchison. Optimal use of experience in first person shooter environments. In *2019 IEEE Conference on Games (CoG)*, pages 1–4. IEEE, 2019.

[2] 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*, 2020.

[3] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[4] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[7] P. Cichosz. Truncating temporal differences: On the efficient implementation of td (lambda) for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318, 1994.

[8] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[9] K. W. Cobbe, J. Hilton, O. Klimov, and J. Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

[10] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[11] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[12] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI fall symposium series*, 2015.

[13] M. Hessel, I. Danihelka, F. Viola, A. Guez, S. Schmitt, L. Sifre, T. Weber, D. Silver, and H. Van Hasselt. Muesli: Combining improvements in policy optimization. In *International Conference on Machine Learning*, pages 4214–4226. PMLR, 2021.

[14] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[15] G. Hinton, O. Vinyals, J. Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[16] M. J. Kearns and S. P. Singh. Bias-variance error bounds for temporal difference updates. In *COLT*, pages 142–147. Citeseer, 2000.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[19] S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[22] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[23] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[24] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[27] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[28] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[29] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[30] S. Wu, H. R. Zhang, and C. Ré. Understanding and improving information transfer in multi-task learning. In *International Conference on Learning Representations*, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes] See Section 7.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 7.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes]

(b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix H.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [N/A]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]