DNA: Proximal Policy Optimization with a Dual Network Architecture

Matthew Aitchison
The Australian National University
matthew.aitchison@anu.edu.au

Penny Sweetser
The Australian National University

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 the two tasks. Instead, we show that learning these tasks independently, but with a constrained distillation phase, significantly improves performance. Furthermore, we find that policy gradient noise levels decrease when using a lower *variance* return estimate. Whereas, 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 [32], Atari games [25], and robotics tasks [31]. 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. Until recently, PG approaches [24, 31, 11], while strong on continuous control problems, have under-performed Q-learning approaches on vision-based tasks [25, 16, 4], 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 shared features 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) [31], called Dual Network Architecture (DNA). We test DNA empirically on a subset of the Arcade Learning Environment (ALE) [6], Atari-5 [2]. Our results show a strong increase in performance on this benchmark compared to both PPO and another dual network model Phasic Policy Gradient (PPG) [11]. Our model also outperforms Rainbow DQN [16] on four of the five games tested, even under more challenging environmental settings.

[1] For simplicity, we refer to actor-critic policy gradient as policy gradient, even though some early policy gradient approaches, such as REINFORCE [36], do not learn a value function.

[2] PPO [31], A3C [24], and Muslie [15] all use a joint network when training on vision-based discrete action environments. PPG [11] uses a dual network with an auxiliary task, which we discuss in more detail in Section 2

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

We summarize our contributions. First, we provide empirical results showing an order-of-magnitude difference in the noise scale between the policy gradient and the value loss gradient. 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

Proximal Policy Optimization Our algorithm builds on the well-established Proximal Policy Optimization (PPO) algorithm [31]. PPO is a policy gradient, reinforcement learning algorithm, with many of the advantages of its predecessor Trust Region Policy Optimization [29], while being much simpler to implement. Various aspects of PPO have been studied in recent years, such as the impact of implementation choices [12, 19], the ability to generalize [11, 27], and performance under multi-agent settings [37]. There has also been a growing understanding that the value and policy learning tasks involved in actor-critic models like PPO have important asymmetries [27, 11]. Our work builds on this by investigating how differences in value and policy noise levels can be accommodated to improve PPO's performance on video-based discrete action tasks.

TD(λ) Return Estimation Policy gradient algorithms often make use of a value estimate as a baseline [24], as well as for estimating the value of truncated trajectories [9]. 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) [33]. 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.[3] For a more thorough discussion on this topic see [20].

Noise Scale. Noise scale is a measure of noise-to-signal 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 [23] 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) [11]. 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. This significantly reduces the memory requirements of our algorithm. Second, we make use of a distillation phase rather than an auxiliary phase.[4] Finally, our work reduces gradient noise by using two different return estimators calibrated for the characteristics of each task. We assess the impact of these differences on the performance of the agent in Section 6.

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

[4] 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, it 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 policy gradient and value learning tasks and show that the 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 much 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.[5] 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.[6] 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.

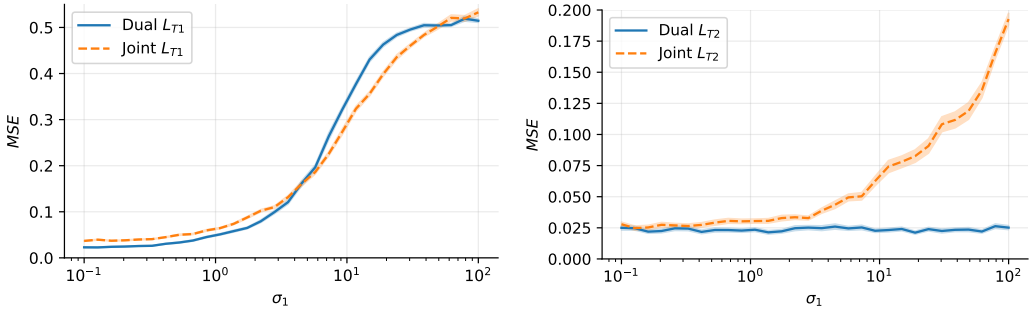

Figure 1: A depiction of the problem of destructive interference on a toy problem. L_{T1} and L_{T2} refer to the mean-squared-error on the first and second task respectively. Shading indicates standard error over 100 seeds.

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 calculating an estimate of the noise scale, developed by [23] 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. 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 [23] have shown, a low bias 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

[5] 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.

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

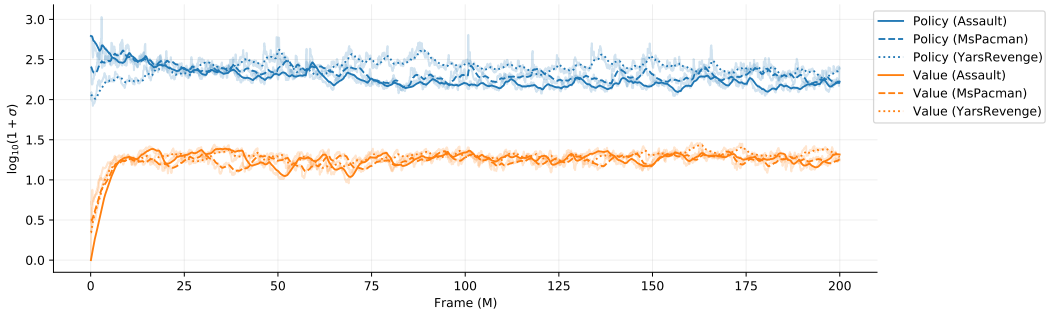

Figure 2: Noise scale for the two learning objectives. For clarity all values are further smoothed using an exponential rolling average (with non-smoothed values presented in a lighter shade). Smoothing may introduce bias, as while $|\mathcal{G}|^2$ and $\mathcal{S}$ are unbiased estimators, their ratio 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 [2]. Like [23], we found it necessary to smooth out noise by maintaining an exponential moving average of $|\vec{G}_{B_{\text{big}}}|^2$. We used $B_{\text{small}} = 32$ and $B_{\text{big}} = 16,384$ in all our experiments. Pseudocode for this procedure is given in Appendix N.

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. 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 each task. Finally, a constrained distillation phase should be applied to take advantage of any constructive interference between the two tasks.[7]

4.1 Independent Networks

Important to DNA is the use of a dual network architecture.[8] 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

[7]The source code used to generate the results in this paper is provided in the supplementary material. We also provide an implementation of our algorithm at https://github.com/maitchison/PPO/tree/DNA.

[8]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 [35].

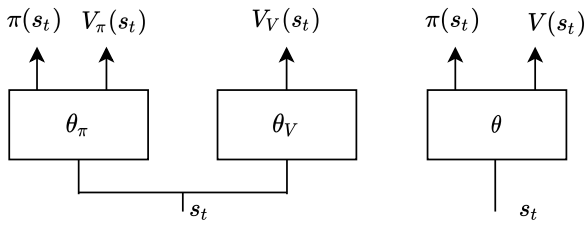

Figure 3: Architecture of DNA (left) compared to a single network setup (right).

calibration of the mini-batch size, which it has been suggested is best set roughly proportional to σ [23]. 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.

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) [33] 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) [30], that is, for any $\lambda \in [0..1], \gamma \in [0..1)$ we have,[9]

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

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 $\lambda_V = \lambda_\pi$.[10] We hypothesize that due to bootstrapping, estimates used for V_{targ} would benefit from being *low bias*, conversely, because estimates used for advantage estimation contribute to the noise of the (already very noisy) policy gradient, they would benefit from being *low variance*. That is, the bootstrapping process may cause symmetrically distributed error, but not biased error, to cancel out. This suggests setting $\lambda_\pi < \lambda_V$, 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 [17]. 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,[11] the process is also very low noise, which we verify in Appendix D.

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

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

[11] 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 $\mathbb{E}[R(s_t)]$.

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 $\frac{1}{N}\sum_{i=0}^{N-1} V_{\text{targ}}^{\pi_{-i}}$ 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.[12]

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 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

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

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) [11], DNA splits training into three distinct phases (policy, value, and distil), 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 phase of DNA is the clipped surrogate object from PPO [31] including the entropy bonus

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

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 the value phase, we we use the squared-error value 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 to handle the high noise of policy learning and a separate network 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.

[12] 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.

Algorithm 1 Proximal Policy Optimization with Dual Network Architecture

procedure PPO-DNA
  Input $N \in \mathbb{Z}^+$ rollout horizon
  Input $A \in \mathbb{Z}^+$ number of agents
  Input π the initial policy.
  for $i = 1$ to ... **do**
    for $a = 1$ to A **do**
      Run policy π in environment a for N timesteps
    Compute $V_{\text{targ}} \leftarrow \text{TD}^{(\gamma,\lambda_V)}$
    Compute $V_{\text{adv}} \leftarrow \text{TD}^{(\gamma,\lambda_\pi)}$
    Compute $\hat{A} \leftarrow \text{TD}^{(\gamma,\lambda_\pi)} - V_{\text{adv}}$
    for $j = 1$ to E_π **do**
      Optimize L^{CLIP} wrt θ_π
    for $j = 1$ to E_V **do**
      Optimize L^{VF} wrt θ_V
    $\pi_{\text{old}} \leftarrow \pi$
    for $j = 1$ to E_D **do**
      Optimize L^D wrt θ_π
  Output π

5 Evaluation

To evaluate our algorithm's performance, we used the Atari-5 benchmark [2]. Scores in Atari-5 are generated using a weighted geometric average over five specific games and produce results that correlate well with the median score if all 57-games had been evaluated. This allowed us to perform multiple seeded runs and defined a clear training and test split between the games. In all cases, we fit hyperparameters to the 3-game validation set and only used the 5-game test set for final evaluations.

We opted for the more difficult stochastic ALE settings recommended as best practice by [22]. However, to better understand our results in the context of prior work, we also provide results under the simpler deterministic settings in Appendix F, and additionally provide for reference a single evaluation on the full 57-game set in Appendix M. 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 value and distillation to be the minimum tested (256), while the optimal mini-batch size for policy was the largest tested (2048). For optimization, we used Adam [21], over the standard 200 million frames.[13] 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 [25]. 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 [4]. 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

[13] That is 50 million environment interactions.

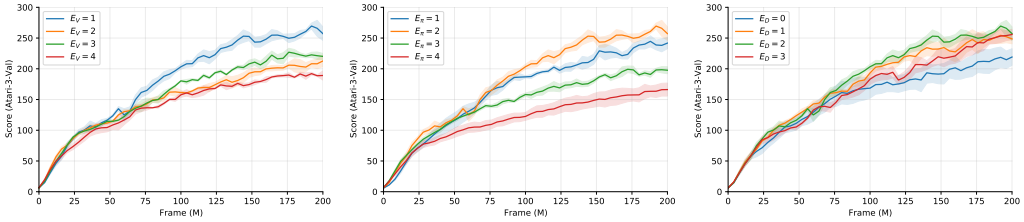

Figure 4: Training curves for DNA with various epochs. The DNA algorithm is most sensitive to the number of policy epochs, and is optimal on our validation set with $E_\pi = 2$. Applying policy updates 3 or 4 times significantly reduces the agents performance.

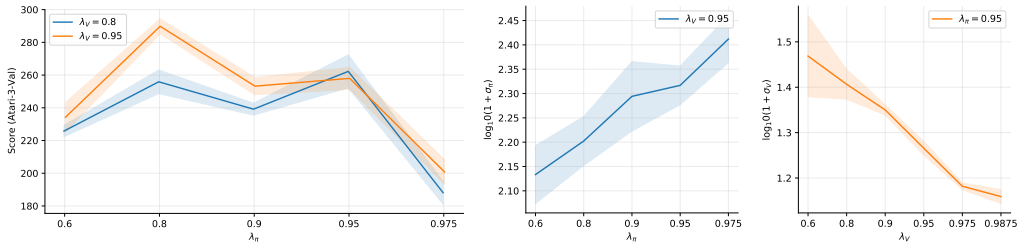

Figure 5: Performance of DNA across a range of λ values for return estimations. Left: The optimal settings for λ_V and λ_π differed. Shading indicates standard error over 5 seeds. Mid/right: Noise levels from a single seed, σ_π increased with λ_π where as σ_V decreased with λ_V. Shading indicates standard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

PPO we also tested against the settings used by [31] 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 DNA to PPO and PPG on Atari-5. We also include supplementary experiments on MuJoCo [34] and Procgen [10] in Appendix K, and L respectively.

6.1 Impact of Training Epochs

We start by investigating the impact of the E_π, E_V, and E_D hyperparameters. 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 (we discuss this further in Section 7). The training curves also suggest 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 that 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 (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 policy gradient noise can be reduced by selecting a low value for λ_π. However, against our expectations, setting λ_V lower actually increased, not decreased the noise level of the value learning task. Once we

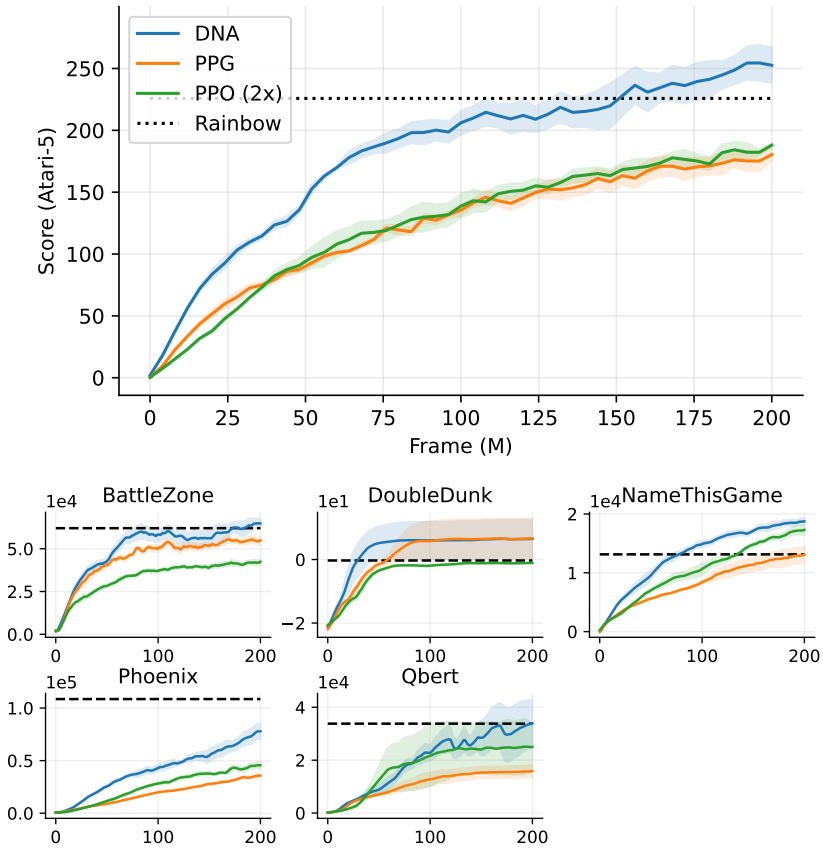

Figure 6: Top: Results on the Atari-5 benchmark. DNA outperforms PPO by a wide margin, and even exceeds Rainbow DQN despite the more difficult settings used in our experiment. Bottom: Individual training curves for each of the five games.

discovered this we ran addition experiments for a broader choice of λ_V, and include the results in Figure 5.

6.3 Comparison 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 scores, as well as training plots for each individual game. We also include references for Rainbow DQN, although we note that these scores were generated under the simpler deterministic settings.

7 Discussion

Reduced Training as an Alternative to Experience Replay We found that *less* training, both on policy and value updates, typically improves our agent's performance. One explanation for this could be, as [11] have suggested, that in the case of value learning, this could 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 a large change in the policy, which may suggest a better constraint than PPO's clipping is required.[15] That being said, limiting the training to one or two epochs provides straightforward solution to the problem while having the additional benefit of decreasing the computational resources required to train an agent.

[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.

Choice of λ Our results give 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* values of λ_V. That is, policy learning prefers low *variance*, but value learning prefers low *bias*. This result may explain why clipping, which introduces bias, performs poorly on value learning but quite well on policy learning.[16] Our results also suggest that an even higher choice of λ_V than the 0.95 we used in our experiments may be appropriate.

Update-to-Data Ratio Our work also raises an interesting question about the update-to-data (UTD) ratio differences between value and policy learning. We found DNA to work best with only a single value network update but four policy network updates. This asymmetry could be due to the extremely high noise levels for policy updates. The single value update contrasts with other work where much higher value learning UTD ratios was found to be preferred [1, 8].

Limitations Our paper's primary focus was improving PG results on discrete action vision-based problems, as this is where PG has traditionally unperformed Deep Q-learning. However, we do also provide an initial look into continuous control problems on the MuJoCo [34] dataset in Appendix K.

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, which 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 [14], distributional value learning [5], improved feature encoders [13], better exploration bonuses [7], and making use of a learned model [28].

8 Conclusion

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 the training of 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.

Acknowledgments and Disclosure of Funding

This research was supported by an Australian Government Research Training Program (RTP) Scholarship.

[16] By value clipping, we mean the value trust-region suggested by [30], but which has been shown hurt performance by [3].

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. Aitchison, P. Sweetser, and M. Hutter. Atari-5: Distilling the arcade learning environment down to five games. *arXiv preprint arXiv:2210.02019*, 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*, 2020.

[4] 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.

[5] 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.

[6] 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.

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

[8] X. Chen, C. Wang, Z. Zhou, and K. Ross. Randomized ensembled double q-learning: Learning fast without a model. *arXiv preprint arXiv:2101.05982*, 2021.

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[19] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

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

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

[22] 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.

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

[24] 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.

[25] 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.

[26] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054. PMLR, 2018.

[27] R. Raileanu and R. Fergus. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 8787–8798. PMLR, 2021.

[28] 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.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[30] 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.

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

[32] 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.

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

[34] 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.

[35] 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.

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

[37] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

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]

A Implementation Details

The implementation details often matter with reinforcement learning [12]. For this reason, full source code is provided in the supplementary material for all experiments. This section details some of the more important implementation decisions made, most of which match those found in [3]. We ran all experiments (PPO, DNA, PPG) using these same implementation choices to confirm that performance differences were not due differences in implementation.

Reward normalization We normalized rewards such that returns have unit variance, as is common with PPO.[17] Even though we used two separate models, and therefore had less reason to balance the magnitude of the value and policy loss, we still kept reward normalization so that distillation loss, and its interaction with the policy constraint, would be of a similar scale between environments.

Observation normalization We also adopted observation normalization. Each state s was normalized by $s' = \text{clip}((s - s_\mu)/s_\sigma, -3, 3)$ where s_μ was the element wide mean over states seen by the agent so far, and s_σ was the standard deviation. Normalization constants were shared between the policy and value networks.

Repeat action penalty We found that our policy would occasionally get stuck repeating a single action, causing the game to freeze until the time limit occurred. This could occur if the agent mistakenly thought it would get a slightly negative score for continuing and therefore acted to postpone that reward as long as possible.[18] Q-learning algorithms, such as DQN, which make use of ϵ-greedy, do not experience this problem so long as $\epsilon > 0$. To address this, we implemented a reward penalty of 0.25 (normalized) if the agent repeated the same action more than 100 times. We leave finding a better solution to this problem for future work.[19]

Integrating time and action information We added a watermark to the least recent frame in the 4-frame stack indicating the proportion of time which has occurred as a 'progress bar' as well as markers on each frame indicating which action the agent selected on the previous frame. Inclusion of time is necessary to avoid violation of the Markovian property in time-limited environments [26]. Action indicators were added to allow the agent to understand when it repeated the same action multiple times, which is not always possible to determine from the state itself (due to multiple actions causing identical outcomes).

Warmup / desyncing environments When initializing our environments, we ran each of the parallel 128 environments for $t \sim U(1, 1000)$ interactions with actions selected uniformly over the action space. This served two purposes: to provide initial normalization parameters and to desynchronize the environments. We found that if we did not do this, agents would terminate around the same time on some environments, causing parallel rollouts to become correlated. While we found this made very little difference to the agent's performance, it removed oscillating scoring artefacts found early in training in some environments.

B Hyperparameters and Environmental Settings

We selected initial hyperparameters from an initial coarse hyperparameter search on the Atari-3 validation set. In some cases, where only small differences in performances were observed, we prefered settings that had been used in previous papers or were likely to be more efficient. For example, our search found a mini-batch size of 256 optimal for value and distil updates. However, we selected 512 instead, as the difference was not large and found this a more computationally efficient mini-batch size when trained on a GPU. A full list of hyperparameters are given in Table 1. We also provide hyperparameters for our PPG experiments in Table 2. The environmental settings we used are given in Table 3.

[17] We also clipped normalized rewards to $[-5, 5]$ but found that this occurred exceedingly rarely, especially after the first 1 million frames.

[18] An example of this would be the agent failing to press the reset button after losing a life in Breakout.

[19] Adding some kind of exploration strategy, such as Random Network Distillation [7] would likely solve this problem more elegantly.

Setting	DNA	PPO	PPO_{orig}
Entropy bonus (c_{eb})	0.01	0.01	$\alpha \times 0.01$
Rollout horizon (N)	128	128	128
Parallel agents (A)	128	128	8
PPO epsilon ϵ	0.2	0.2	0.1
Discount gamma (γ)	0.999	0.999	0.99
Learning Rate	2.5×10^{-4}	2.5×10^{-4}	$\alpha \times 2.5 \times 10^{-4}$
Policy lambda (λ_π)	0.95	0.95	0.95
Value lambda (λ_V)	0.95	0.95	0.95
Policy epochs (E_π/E_{ppo})	2	2	3
Value epochs (E_V)	2	-	-
Distil epochs (E_D)	2	-	-
Distil beta (β)	1.0	-	-
Policy mini-batch size	2048	2048	256
Value mini-batch size	512	-	-
Distil mini-batch size	512	-	-
Repeated action penalty	0.25	0.25	0.25
Global gradient clipping	5.0	5.0	5.0

Table 1: Summary of hyperparameters found in coarse hyperparameter search. Epoch counts, and λ_* values were further fine-tuned as detailed in the main study. For PPO_{orig}, α was linearly annealed over training from $[1, 0]$.

Setting	PPG	PPG (tuned)
Policy epochs (E_π)	1	2
Value epochs (E_V)	1	1
Distil epochs (E_D)	0	0
Auxiliary epochs (E_{aux})	6	2
Auxiliary Period (N_π)	32	32

Table 2: Summary of hyperparameters used in the Phasic Policy Gradient experiments. All other hyperparameters were set according to the DNA settings in Table 1.

Setting	Easy	Hard
Terminal on Loss of Life	True	False
Action Space	Minimal	Full
Repeat Action Probability	0.0	0.25
Training frames	200M	
Color / grayscale	Grayscale	
Frame stacked	4	
Action repetitions	4	
Reward clipping	No	
Episode timeout	108K	
Resolution	84×84	
Noop Starts	1-30	

Table 3: Environmental Settings used in experiments. 'Hard' mode settings follow best practice by [22], 'easy' mode correspond to those used in the Rainbow DQN paper [16], with the exception that we do not apply the domain specific reward clipping modification. Training frames includes skipped frames, that is our agents performed 50M interactions with the environment.

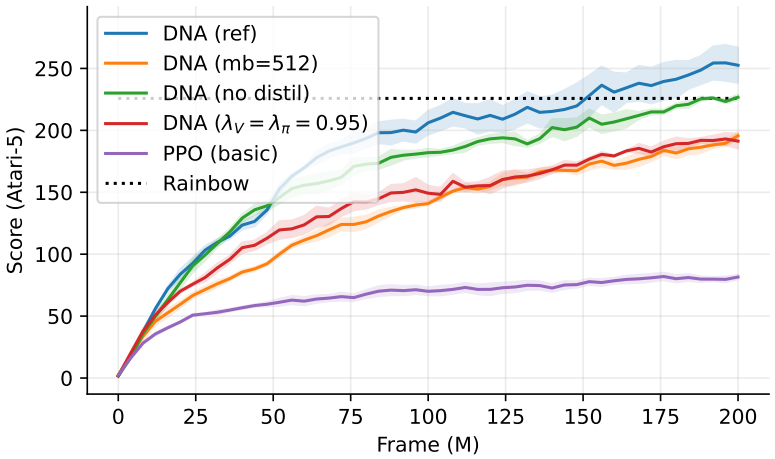

Figure 7: Training curves for the ablation studies. Shading indicates standard error over 3 seeds.

C Ablation Study

This appendix quantifies the performance contribution of several important components of DNA. We considered the following changes:

- **DNA (mb=512)** Our analysis of noise levels suggested a much larger mini-batch size for policy updates than for value updates. We measure the impact of this change by evaluating DNA with mini-batch sizes for all three training objectives set to 512.
- **DNA (no distil)** Validation scores indicated an improvement in performance using distillation over no distillation. We verify that this result is replicated on our test set.
- **DNA ($\lambda_V = \lambda_\pi = 0.95$)** We measured the impact of using non-homogeneous values for λ_V and λ_π by testing with these both set to 0.95.

We also include the reference run from the main study and a "PPO (basic)" run, which was a single network with the 'Nature-CNN' encoder, and a mini-batch size of 512, and can be thought of as DNA with all novel components turned off. Results are provided in Figure 7, along with the score, and performance regression in Table 4. We found non-homogeneous values for λ_V and λ_π, and a larger policy mini-batch size, to be the most significant changes, with distillation also providing some benefit.

Table 4: Atari-5 scores for each of the ablation runs.

Run	Atari-5 Score	Regression
DNA (ref)	252	0.0
DNA (no distil)	226	-10.2%
DNA (mb=512)	195	-22.5%
DNA ($\lambda_V = \lambda_\pi = 0.95$)	191	-24.3%
PPO (basic)	81	-67.7%

D Noise Scale for Distillation Learning

In this appendix, we present the noise scale results for distillation learning. We expected distillation to have a low noise level because the targets are drawn from the relatively noise-free value network estimations and not from the higher variance value targets. Our results in Figure 8 confirm this hypothesis. Of note is that the difference in noise scale between environments was much more significant for distillation loss than it was for the value or policy loss. These results indicate that distillation may benefit from smaller mini-batch sizes. However, the decreased efficiency of processing these smaller batches on GPU hardware may out-weigh any potential advantages.

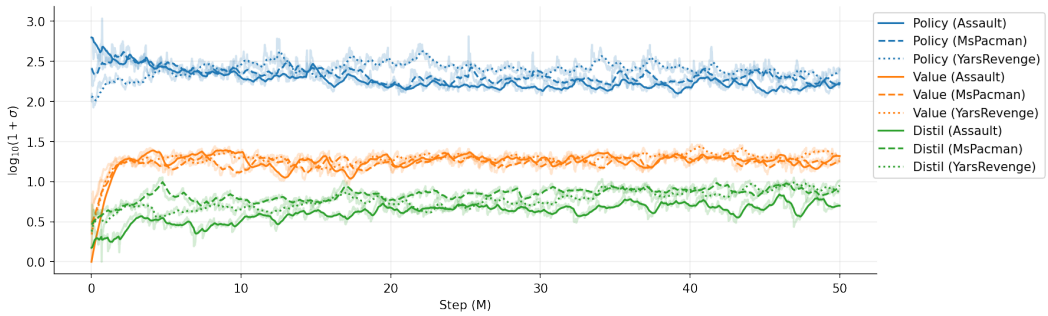

Figure 8: Noise level of the three tasks, policy learning, value learning, and distillation, over the three games in our validation set.

E Proof of Relationship between GAE and TD(λ)

We provide a proof that calculating the General Advantage Estimate [30] is equivalent to calculating TD(λ) returns, then subtracting the state value estimate. Concretely, for some $\lambda \in [0..1)$ and $\gamma \in [0..1]$ we have from [30]

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + \Big(\sum_{i=0}^{k-1} \gamma^i r_{t+i} \Big) + \gamma^k V(s_{t+k}) \tag{13}$$

$$= -V(s_t) + \text{NSTEP}^{(\gamma,k)}(s_t). \tag{14}$$

The GAE advantage estimate is defined as an exponentially weighted sum of these $A^{(k)}$'s as follows

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} := \sum_{i=0}^{\infty} (1-\lambda)\lambda^i \hat{A}_t^{(i+1)} \tag{15}$$

$$= (1-\lambda) \sum_{i=0}^{\infty} \lambda^i (-V(s_t) + \text{NSTEP}^{(i+1)}(s_t)) \tag{16}$$

$$= \left[(1-\lambda) \sum_{i=0}^{\infty} \lambda^i (-V(s_t)) \right] + \left[(1-\lambda) \sum_{i=0}^{\infty} \lambda^i \text{NSTEP}^{(i)}(s_t) \right] \tag{17}$$

$$= -V(s_t) + \text{TD}^{(\gamma,\lambda)}(s_t) \tag{18}$$

as required.

F Results under Rainbow DQN Style Environmental Settings.

In our main study, we compared DNA to PPO on the Atari-5 benchmark under the recommended settings given by [22]. Many prior results have been generated using the simpler, non-stochastic

version of the environments and with domain-specific knowledge, such as loss of life as a terminal state and a custom clipped reward modifier. We evaluated DNA and our implementation of PPO here in the simplified environments and found a modest improvement under these settings. We provide these results for better comparison against previous works.

In these experiments we did not use reward clipping. Reward clipping is a domain-specific reward modification that reduces all positive rewards to +1 and all negative rewards to -1. We were concerned that clipping rewards would bias the algorithm, as the agent is optimizing a reward structure that may not match the true rewards of the game. It could be that reward clipping may be necessary for Deep Q-learning approaches to reduce high variance returns.[20] However, we have not found this to be an advantage over reward normalization for PPO or DNA.

We found that DNA outperformed Rainbow DQN on all five environments, and obtained a better Atari-5 score after just 49M environmental frames. Proximal Policy Optimization, with a single policy update, also outperformed Rainbow DQN on this task.

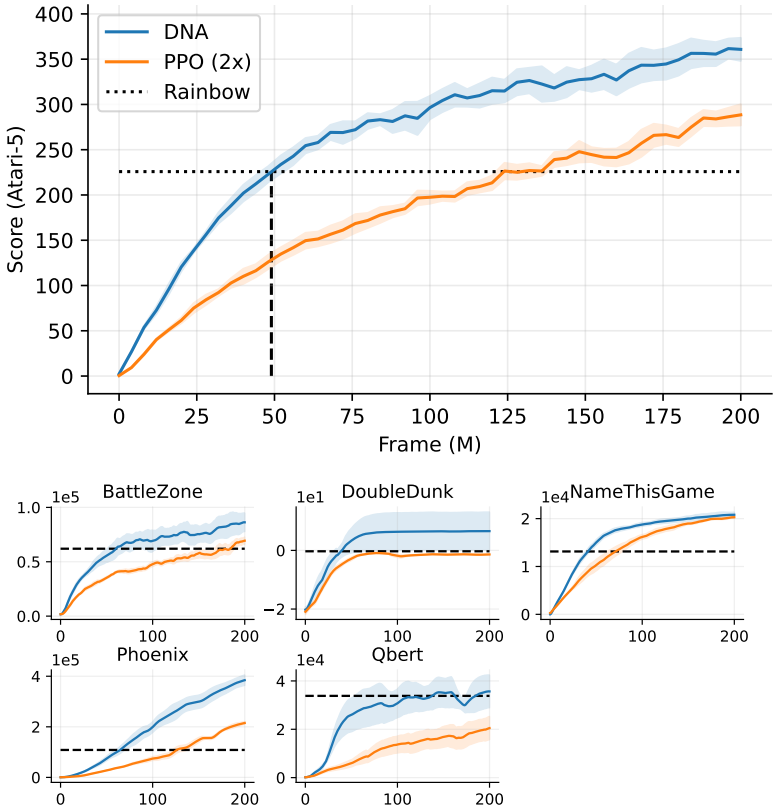

Figure 9: Results on the Atari-5 benchmark, with 'easy' environmental settings matched to those used by Rainbow DQN [16]. Shaded regions indicate standard error over three seeds.

G Supplementary Results

We investigated some supplementary questions. Specifically, we wanted to validate that the performance improvement of DNA compared to PPO and PPG did not result solely from the hyperparameter choices. We, therefore, evaluated PPO and PPG under a range of alternative hyperparameters on Atari-5 and note the results here.

We tested a variety of alternatives for PPO as described below. We found that none of the alternative settings resulted in improved performance (Figure 10 *left*).

[20] Or squashing the value function, see Appendix A of [4].

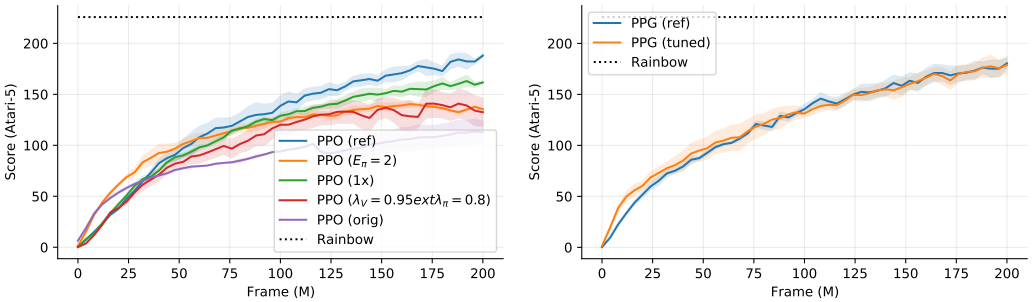

Figure 10: Training curves for the supplementary studies. Shading indicates standard error over 3 seeds. Reference runs are from the main study.

- **PPO** ($E_\pi = 2$) Our tuning process found a single epoch optimal for PPO, but two epochs optimal for DNA, would PPO have performed better if it was given two epochs instead of one? We found that while initial performance was stronger, this change ultimately regressed the performance.
- **PPO** ($\lambda_V = 0.95\ \lambda_\pi = 0.8$) Using separate return estimations for advantages and value targets does not require a dual network setup. Therefore we checked if the performance of PPO can be improved by using the non-homogeneous λ values used in our DNA experiment. We found these settings regressed performance.
- **PPO (1x)** In our experiments DNA and PPO used different network encoders (PPO used twice as many channels). It is possible that increasing the parameters made training more difficult for PPO? We tested PPO with the standard NatureCNN encoder, and found this change regressed performance.
- **PPO (orig)** Our settings for PPO deviated from those used by [31]. For completeness we also provide results using these settings. We found that while the these original settings performed well initially, they eventually unperformed the reference by a large margin. We also note that these settings took much longer to train (see Appendix H).

We also evaluated PPG with alternative settings (Figure 10 *right*).

- **PPG (tuned)** In our main experiment we evaluated PPG using $E_\pi = 1, E_V = 1, E_{\text{aux}} = 6$ taken from [11]. These differ significantly from those used for DNA. We therefore reevaluated PPG using $E_\pi = 2, E_V = 1, E_{\text{aux}} = 2$ which more closely match the settings used by DNA. We found this change had little impact on the performance of the algorithm.

H Training Time

DNA makes use of two independent networks and three training phases, which may have a negative effect on training time. We examine this here. All times are approximate and for comparative purposes only. Rainbow DQN times are on different hardware and using a different codebase.

Our implementation of DNA ran very quickly and is faster than PPO when PPO is configured as per [31]. This is due to our use of more parallel agents (128 vs 8) coupled with a larger mini-batch size. We give approximate training times in Table 5 which were taken from a 24-core machine with four 2080-TIs. We found we could train 8 DNA models in 8-hours on our 4-GPU machine, giving a rate of 4 GPU hours per game learned.

I Tuning for Proximal Policy Optimization

We repeated the same hyperparameter sweep on PPO as we did for DNA for a fair comparison. We found that, like DNA, PPO also benefited greatly from reduced epochs during training. We present

[21] There are faster ways of training DQN like algorithms, for example Ape-X [18].

Table 5: Approximate training times for the algorithms used in this paper.

Algorithm	GPU hours per game
PPO (our settings)	3
DNA	4
PPG	4.5
PPO ([31] settings)	7.5
Rainbow DQN	240[21]

the results here for 1,2,3 and 4 epochs, along with a search over the choice for λ used in the General Advantage Estimate. Results from the main study used the best performing model, which was found to be $\lambda = 0.95$, and $\mathrm{E}_\pi = 1$. These hyperparameters differ from those used by [31], and achieve a significant improvement in performance with less computation (see Appendix F, H).

We found the performance of PPO to plateau after a point in training which decreases with the number of training epochs. This is consistent with [31], who trained for 40M frames and whose results show performance levelling off around 20M. However, when fewer epochs are used, performance continues to increase after these points. We also performed some quick experiments using partial epochs (0.5 and 0.75) but found these under-performed a single epoch and have not included the results here. We also found that the single network setup of PPO did not benefit as much as DNA from tuning the λ parameter and that the commonly used $\lambda = 0.95$ was optimal for our training set.

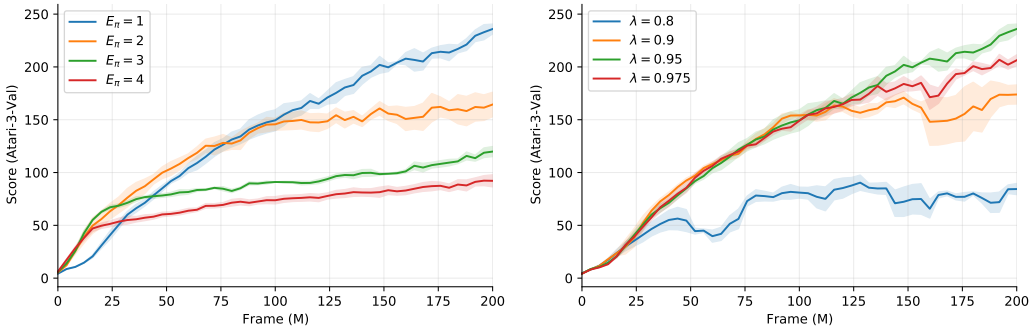

Figure 11: Training Curves for Proximal Policy Optimization over various epoch counts, and settings for λ_{GAE}. Shaded area indicates standard error over three seeds.

J Distillation Targets

We evaluated two distillation targets for our distillation phase: a *random projection* into $\mathbb{R}^{16}$, and the value networks *value estimates* as well as a third strategy, *feature matching*.[22] Random projection and value estimate outperformed the no distillation baseline, and feature matching underperformed the baseline. Results (Figure 12) are from a single seed on the Atari-3 validation dataset. In all cases, distillation was trained on two passes of trajectories sampled from the rollout and used a policy constraint.

K Additional Results on MuJoCo

We applied DNA to the robotics task MuJoCo [34]. We used hyperparameters based on the work of [31]. However, we used the slightly different 'v2' versions of the environments rather than the 'v1' versions used in their experiments. Scores are taken during training. PPO and DNA learned a per action standard deviation independent of the state.

[22] That is, the distillation step minimized the mean squared error between the features outputted by the policy network and the features output by the value network. As with our other targets, gradients were only propagated through the policy network.

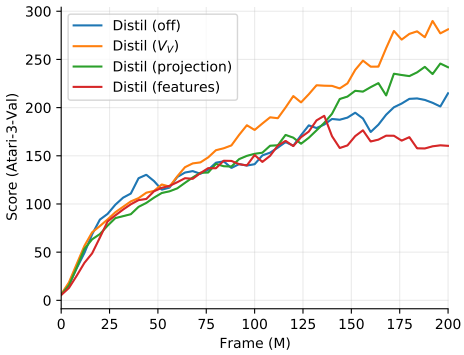

Figure 12: Performance of the four distillation strategies trailed. Only a single seed was used.

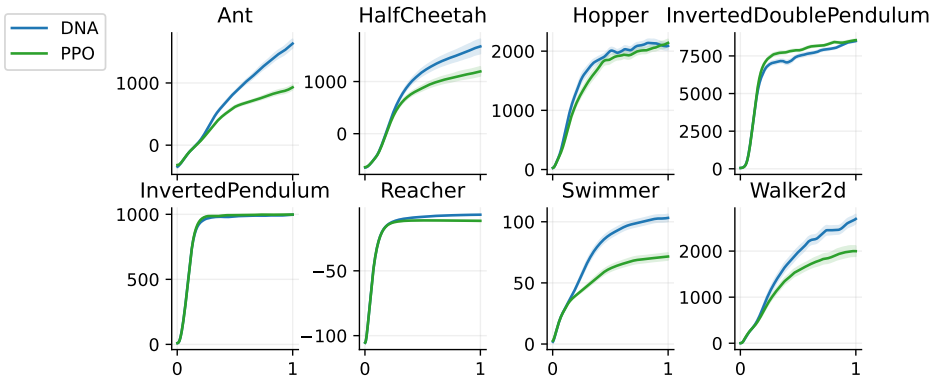

Figure 13: Results from the MuJoCo experiments for PPO and DNA over 30 seeds, with one standard error shown shaded.

DNA shows an improvement over PPO on five of the eight of the environments tested (*Ant*, *HalfCheetah*, *Reacher*, *Swimmer*, and *Walker2d*) (see Figure 13). In the remaining three environments, DNA and PPO produce similar results. We performed only basic hyperparameter tuning, using the *Walker2D* environment. Hyperparameters for these experiments are given in Table 6.

Unlike in our Atari experiments, we found setting λ_π to 0.8 to produce poor results, and so reverted this setting back to 0.95. We removed the KL divergence penalty for distillation and replaced it with the mean-squared error between the center of the Gaussian distribution outputs for the policy and value networks.

Table 6: Hyperparameters used for MuJoCo. Hyperparameters follow closely to that of [31]. Agents where trained for one-million interactions. † Learning rate was annealed linearly to 0.0 over training.

Setting	PPO	DNA
Entropy bonus (c_{eb})	0.01	
Rollout horizon (N)	2048	
Parallel agents (A)	1	
PPO epsilon ϵ	0.2	
Discount gamma (γ)	0.99	
Learning Rate	3.0×10^{-4}†	
Policy lambda (λ_π)	0.95	
Value lambda (λ_V)	0.95	
Policy epochs (E_π / E_{ppo})	10	10
Value epochs (E_V)	-	10
Distil epochs (E_D)	-	10
Distil beta (β)	-	1.0
Policy mini-batch size	64	64
Value mini-batch size	-	64
Distil/Aux mini-batch size	-	64
Global gradient clipping	5.0	5.0

L Additional Results on Procgen

We ran additional experiments on the Procgen benchmark [10]. Each environment within this benchmark requires learning across a set of 200 procedurally generated environments. Because of this, algorithms benefit from large replay buffers able to capture the diverse conditions under which the agent must act. Unlike PPG, DNA does not make use of replay buffer and so is not likely to perform well at this task.

Despite this, we found DNA to be competitive with PPG on many of the environments tested (*coinrun*, *fruitbot*, *leaper*, *maze*, *miner*, *ninja*, *dodgeball* and *heist*), as shown in Figure 15. When scores across all environments are normalized, DNA produces an average normalized score of 0.65 compared to 0.49 for PPO and 0.75 for PPG (see Figure 14).

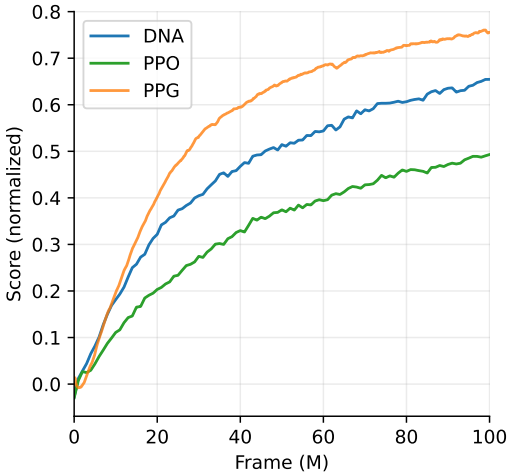

Figure 14: Results on the Procgen benchmark (hard distribution settings). Scores are average normalized score over all 16 environments. PPG results are taken from [11].

We kept hyperparameters close to that of [11] and used the *bigfish* environment to select one policy, one value, and two distil epochs as optimal for DNA. We note that these settings require the observations to be forwarded through a network only four times, compared to fourteen times in PPG. This makes DNA similar in computational requirements to PPO.[23]

We generated results for DNA and PPO and used the results supplied by [11] for PPG.[24] We also noticed a modest decline in the performance of our implementation of PPO compared to that of [11] (0.49 vs 0.58). This could be due to our use of observation normalization. The difference in performance is especially apparent in the environment *plunder* (for both DNA and PPO). This might indicate that performance could be improved further by adopting the fixed scaling preprocessing procedure used in PPG. We give hyperparameters for our experiments in Table 7.

[23] PPG's auxiliary phase requires forwarding through both the policy and value networks, whereas DNA's distillation update only requires a forward through the policy network.

[24] As recorded in the corresponding Github repository `https://github.com/openai/procgen`. We note some artefacts in the scores they supply, most notable in *heist* and *maze*. We believe this may be due to how their environments were initialized.

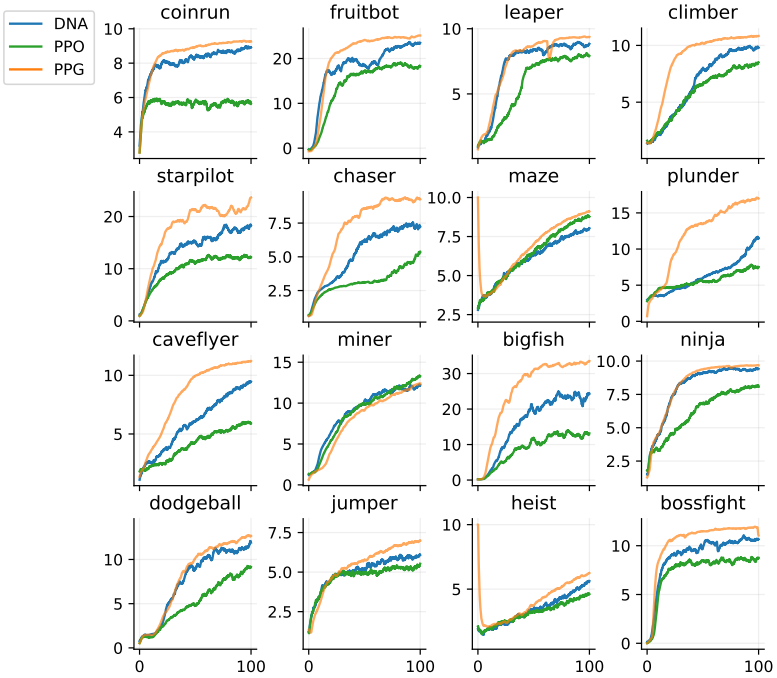

Figure 15: Results on each game in the Procgen benchmark (hard distribution settings). Scores smoothed for clarity. PPG results are taken from [11] and are an average over three seeds. PPO and DNA are our results over one seed.

Table 7: Hyperparameters used for ProcGen. PPG hyperparameters are taken from [11].

Setting	PPO	DNA	PPG
Entropy bonus (c_{eb})		0.01	
Rollout horizon (N)		256	
Parallel agents (A)		256	
PPO epsilon ϵ		0.2	
Discount gamma (γ)		0.999	
Learning Rate		5.0×10^{-4}	
Policy lambda (λ_π)		0.95	
Value lambda (λ_V)		0.95	
Repeated action penalty		0.0	
Policy epochs (E_π / E_{ppo})	3	1	1
Value epochs (E_V)	-	1	1
Distil/Aux epochs (E_D)	-	2	6
Distil/Aux beta (β)	-	1.0	1.0
Policy mini-batch size	8192	8192	8192
Value mini-batch size	-	2048	8192
Distil/Aux mini-batch size	-	512	4096
Global gradient clipping	5.0	5.0	off

M Additional Results on ALE

Our main study used Atari-5 to allow enough time for seeded runs and because it provides an established training/test split. Because Atari-5 is a new benchmark, we thought it important to validate our algorithm's performance on the full 57-game suite. We, therefore, provide results for both PPO (2x) and DNA on all 57 games in the ALE using both the 'easy' settings similar to [16] and the more difficult settings used in our main study.

We measured the median score as the median human-normalized score over the past 100-episodes and report the final median scores as the average for this measure over the last 10M frames (5% of training frames). Individual game scores are also reported as the average over the final 10M frames.

We found, under both the hard and the easy settings, DNA outperformed PPO by a wide margin (see Table 8). DNA also outperformed Rainbow DQN on the easy settings after just 85.5M training frames (Figure 16). Training plots are provided in Figures 17, 18. Results for each game are given in Tables 9, 10. Most surprising is that when trained with only a single policy epoch, a larger batch size, and more parallel agents, PPO becomes comparable to Rainbow DQN on the easy settings despite being a much simpler algorithm and being 80-times faster to train.

Table 8: Summary of results on the Atari-57 benchmark.

Algorithm	Median (easy)	Median (hard)
Rainbow	223	-
PPO	224	155
DNA (ours)	**311**	**207**

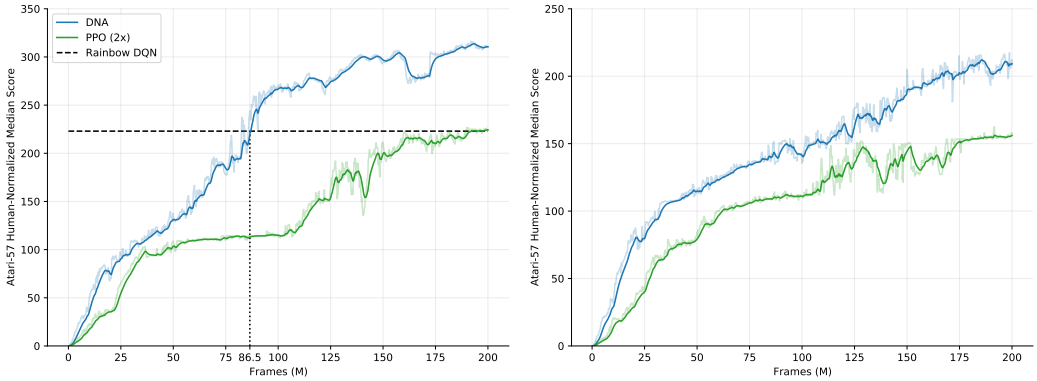

Figure 16: Median Score over all 57 games for DNA and PPO with 2x parameters. Left: performance under easy settings. DNA matches Rainbow DQN performance after just 86.5M frames. Right: performance under hard settings.

N Pseudocode for Generating Noise Scale Estimates.

In their paper, McCandlish et al. [23] define the noise scale as,

$$\mathcal{B}_{\text{noise}} := \frac{\text{tr}(H\Sigma)}{G^T H G} \tag{19}$$

where at parameter values θ, G is the true gradient, H is the true Hessian, and Σ is the per-example covariance matrix. For large models, the calculation of the Hessian is not practical. We, therefore, use their simplified measure

$$\mathcal{B}_{\text{simple}} := \frac{\text{tr}(\Sigma)}{|\vec{G}|^2}. \tag{20}$$

Even though this formulation makes the unrealistic assumption that the Hessian is a multiple of the identity matrix, empirical studies by McCandlish et al. have shown it to provide a surprisingly good approximation for $\mathcal{B}_{\text{noise}}$.

Our method for generating estimates of $\mathcal{B}_{\text{simple}}$ follows closely that of [23] and is formalized in Algorithm 2. While the estimates for $\text{tr}(\Sigma)$, and $|\vec{G}|^2$ are both unbiased their ratio may not be. We mitigate this by averaging over multiple samples when calculating $\vec{G}_{B_{\text{small}}}$ and applying smoothing to our estimate for $|\mathcal{G}|^2$.

Algorithm 2 Estimate Noise Scale

function ESTIMATENOISESCALE(
  D, a batch of data.
  L, a loss function.
  θ, the model parameters.
  N_{samples}, the number of small mini-batches to use.
  $B_{\text{big}}, B_{\text{small}}$, the big and small mini-batch sizes.
  $|\mathcal{G}|^2_{\text{old}}$, the previous smoothed $|\mathcal{G}|^2$ value.
  α, smoothing factor to use for $|\mathcal{G}|^2$)
  Define SAMPLE(x,y), to draw y samples from x without replacement.
  $D_{\text{big}} \leftarrow \text{SAMPLE}(D, B_{\text{big}})$
  $|\vec{G}_{B_{\text{big}}}|^2 \leftarrow |\nabla_\theta L_{D_{\text{big}}}(\theta)|^2$
  $|\vec{G}_{B_{\text{small}}}|^2 \leftarrow 0$
  for $i = 1..N_{\text{samples}}$ **do**
    $D_{\text{small}} \leftarrow \text{SAMPLE}(D, B_{\text{small}})$
    $|\vec{G}_{B_{\text{small}}}|^2 \leftarrow |\vec{G}_{B_{\text{small}}}|^2 + (1/N_{\text{samples}})|\nabla_\theta L_{D_{\text{small}}}(\theta)|^2$
  $|\mathcal{G}|^2_{\text{new}} \leftarrow (B_{\text{big}}|\vec{G}_{B_{\text{big}}}|^2 - B_{\text{small}}|\vec{G}_{B_{\text{small}}}|^2)/(B_{\text{big}} - B_{\text{small}})$
  $\mathcal{S} \leftarrow (|\vec{G}_{B_{\text{small}}}|^2 - |\vec{G}_{B_{\text{big}}}|^2)/(1/B_{\text{small}} - 1/B_{\text{big}})$
  $|\mathcal{G}|^2 \leftarrow \alpha|\mathcal{G}|^2_{\text{old}} + (1-\alpha)|\mathcal{G}|^2_{\text{new}}$ ▷ Perform smoothing over $|\mathcal{G}|^2$, to reduce variance
  $\mathcal{B}_{\text{simple}} \leftarrow \mathcal{S}/|\mathcal{G}|^2$
  return $\mathcal{B}_{\text{simple}}$ ▷ the (squared) noise scale estimate.
  return $|\mathcal{G}|^2$ ▷ pass to next call of EstimateNoiseScale

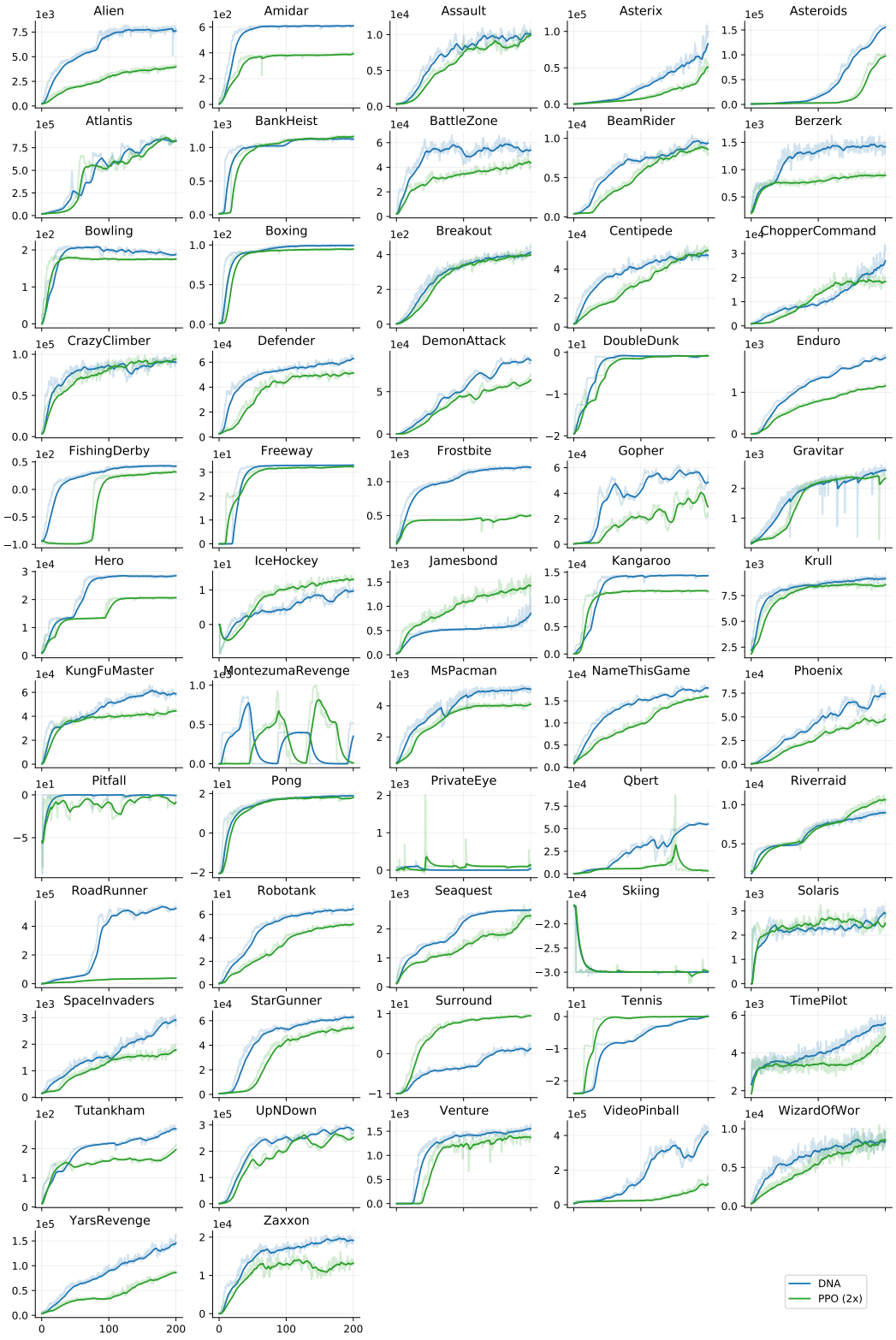

Figure 17: Training plots for DNA on all 57 games in the Atari-57 benchmark under 'hard' settings. Results are from a single seed, with smoothed results in bold, and non-smoothed results shown faded.

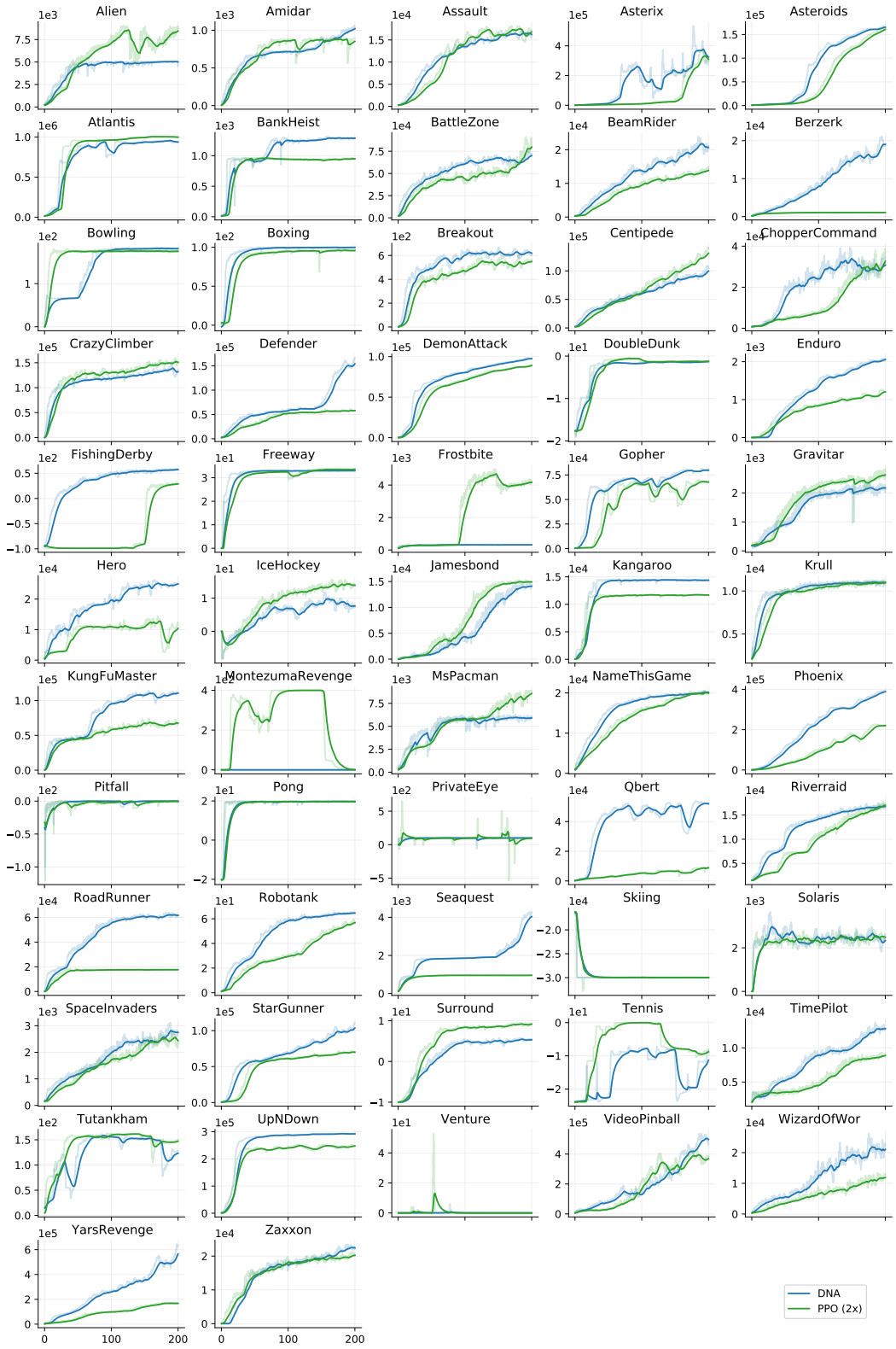

Figure 18: Training plots for DNA on all 57 games in the Atari-57 benchmark under 'easy' settings. Results are from a single seed, with smoothed results in bold, and non-smoothed results shown faded.

Table 9: Final scores for all 57 games in ALE under 'hard' settings. Reported as mean score over the final 10M frames of training.

Game	Random	Human	PPO (2x)	DNA
Alien	228	7,128	3,976	**7,617**
Amidar	5.8	**1,720**	391	610
Assault	222	742	10,098	**10,282**
Asterix	210	8,503	52,958	**85,070**
Asteroids	719	47,389	99,458	**157,926**
Atlantis	12,850	29,028	**816,033**	813,564
BankHeist	14.2	753	**1,159**	1,125
BattleZone	2,360	37,188	44,227	**54,462**
BeamRider	364	**16,926**	8,587	9,369
Berzerk	124	**2,630**	897	1,429
Bowling	23.1	161	175	**187**
Boxing	0.1	12.1	94.7	**99.4**
Breakout	1.7	30.5	399	**416**
Centipede	2,091	12,017	**53,676**	49,444
ChopperCommand	811	7,388	18,160	**26,998**
CrazyClimber	10,780	35,829	**94,695**	89,864
Defender	2,874	18,689	51,002	**62,935**
DemonAttack	152	1,971	64,180	**88,673**
DoubleDunk	-18.6	-16.4	-0.8	**-0.8**
Enduro	0.0	860	1,150	**1,819**
FishingDerby	-91.7	-38.7	32.0	**41.9**
Freeway	0.0	29.6	32.3	**33.0**
Frostbite	65.2	**4,335**	497	1,211
Gopher	258	2,412	28,785	**46,348**
Gravitar	173	**3,351**	2,209	2,627
Hero	1,027	**30,826**	20,673	28,526
IceHockey	-11.2	0.9	**12.9**	9.4
Jamesbond	29.0	303	**1,444**	861
Kangaroo	52.0	3,035	11,556	**14,367**
Krull	1,598	2,666	8,539	**9,161**
KungFuMaster	258	22,736	44,648	**58,895**
MontezumaRevenge	0.0	**4,753**	0.0	385
MsPacman	307	**6,952**	4,102	5,067
NameThisGame	2,292	8,049	16,050	**18,155**
Phoenix	761	7,243	47,804	**75,709**
Pitfall	-229.4	**6,464**	-10.4	-0.7
Pong	-20.7	14.6	17.9	**18.9**
PrivateEye	24.9	**69,571**	129	36.5
Qbert	164	13,455	3,273	**54,706**
Riverraid	1,338	**17,118**	10,642	9,005
RoadRunner	11.5	7,845	38,970	**520,458**
Robotank	2.2	11.9	51.5	**65.0**
Seaquest	68.4	**42,055**	2,494	2,655
Skiing	-17098.1	**-4336.9**	-29553.4	-29974.5
Solaris	1,236	**12,327**	2,379	2,976
SpaceInvaders	148	1,669	1,808	**2,940**
StarGunner	664	10,250	54,488	**62,760**
Surround	-10.0	6.5	**9.5**	0.9
Tennis	-23.8	-8.3	0.0	**0.2**
TimePilot	3,568	5,229	4,907	**5,554**
Tutankham	11.4	168	199	**272**
UpNDown	533	11,693	250,253	**280,014**
Venture	0.0	1,188	1,375	**1,562**
VideoPinball	0.0	17,668	120,142	**432,752**
WizardOfWor	564	4,756	**8,834**	8,480
YarsRevenge	3,093	54,577	87,267	**147,864**
Zaxxon	32.5	9,173	13,244	**19,125**

Table 10: Final scores for all 57 games in ALE under 'easy' settings. Reported as mean score over the final 10M frames of training.

Game	Random	Human	Rainbow DQN	PPO (2x)	DNA
Alien	228	7,128	**9,492**	8,525	5,021
Amidar	5.8	1,720	**5,131**	844	1,025
Assault	222	742	14,198	**16,688**	16,293
Asterix	210	8,503	**428,200**	321,207	323,965
Asteroids	719	47,389	2,713	161,787	**165,973**
Atlantis	12,850	29,028	826,660	**997,292**	932,559
BankHeist	14.2	753	**1,358**	951	1,286
BattleZone	2,360	37,188	62,010	**82,834**	71,003
BeamRider	364	16,926	16,850	13,932	**20,393**
Berzerk	124	2,630	2,546	1,083	**19,789**
Bowling	23.1	161	30.0	175	**181**
Boxing	0.1	12.1	99.6	95.6	**99.9**
Breakout	1.7	30.5	418	553	**626**
Centipede	2,091	12,017	8,167	**131,062**	100,194
ChopperCommand	811	7,388	16,654	**31,912**	31,181
CrazyClimber	10,780	35,829	**168,788**	151,937	131,623
Defender	2,874	18,689	55,105	58,201	**152,768**
DemonAttack	152	1,971	**111,185**	88,958	97,909
DoubleDunk	-18.6	-16.4	**-0.3**	-1.3	-1.3
Enduro	0.0	860	**2,126**	1,230	2,059
FishingDerby	-91.7	-38.7	31.3	29.0	**57.4**
Freeway	0.0	29.6	**34.0**	33.5	33.0
Frostbite	65.2	4,335	**9,590**	4,190	320
Gopher	258	2,412	70,355	67,850	**80,104**
Gravitar	173	**3,351**	1,419	2,632	2,190
Hero	1,027	30,826	**55,887**	11,125	24,904
IceHockey	-11.2	0.9	1.1	**13.8**	7.2
Jamesbond	29.0	303	**19,480**	14,947	14,102
Kangaroo	52.0	3,035	**14,638**	11,687	14,373
Krull	1,598	2,666	8,742	**11,007**	10,956
KungFuMaster	258	22,736	52,181	67,657	**110,962**
MontezumaRevenge	0.0	**4,753**	384	0.0	0.0
MsPacman	307	6,952	5,380	**8,712**	5,894
NameThisGame	2,292	8,049	13,136	20,053	**20,226**
Phoenix	761	7,243	108,529	220,560	**391,085**
Pitfall	-229.4	**6,464**	0.0	-0.5	0.0
Pong	-20.7	14.6	**20.9**	19.6	19.7
PrivateEye	24.9	**69,571**	4,234	99.9	100
Qbert	164	13,455	33,817	8,836	**52,398**
Riverraid	1,338	17,118	**22,500**	17,156	16,789
RoadRunner	11.5	7,845	**62,041**	17,596	61,713
Robotank	2.2	11.9	61.4	56.7	**64.8**
Seaquest	68.4	**42,055**	15,899	957	4,146
Skiing	-17098.1	**-4336.9**	-12957.8	-29974.4	-29974.0
Solaris	1,236	**12,327**	3,560	2,513	2,225
SpaceInvaders	148	1,669	**18,789**	2,497	2,731
StarGunner	664	10,250	**127,029**	70,247	104,125
Surround	-10.0	6.5	**9.7**	9.1	5.3
Tennis	-23.8	-8.3	**0.0**	-8.8	-10.9
TimePilot	3,568	5,229	**12,926**	8,918	12,774
Tutankham	11.4	168	**241**	147	127
UpNDown	533	11,693	103,600	247,994	**291,934**
Venture	0.0	**1,188**	5.5	0.0	0.0
VideoPinball	0.0	17,668	**533,936**	359,099	505,392
WizardOfWor	564	4,756	17,862	11,996	**20,851**
YarsRevenge	3,093	54,577	102,557	166,670	**564,513**
Zaxxon	32.5	9,173	22,210	20,330	**22,588**