Analyzing the Sensitivity to Policy-Value Decoupling in Deep Reinforcement Learning Generalization

Nasik Muhammad Nafi, Raja Farrukh Ali, William Hsu
Department of Computer Science, Kansas State University
Manhattan, KS 66506, USA
{nnafi, rfali, bhsu}@ksu.edu

Abstract

Existence of policy-value representation asymmetry negatively affects the generalization capability of traditional actor-critic architectures that use a shared representation of policy and value. Fully decoupled/separated networks for policy and value avoid overfitting by addressing this representation asymmetry. However, using two separate networks introduces increased computational overhead. Recent work has also shown that partial separation can achieve the same level of generalization in most tasks while reducing this computational overhead. Thus, the questions arise: *Do we really need two separate networks? Is there any particular scenario where only full separation works? Does increasing the degree of separation in a partially separated network help in generalization?* In this work, we attempt to analyze the generalization performance vis-a-vis the extent of decoupling of the policy and value networks. We compare four different degrees of network separation, namely: fully shared, early separation, late separation, and full separation on the RL generalization benchmark Procgen, a suite of 16 procedurally-generated environments. We show that unless the environment has a distinct or explicit source of value estimation, partial late separation can easily capture the necessary policy-value representation asymmetry and achieve better generalization performance in unseen scenarios, however, early separation fails to produce good results.

1 Introduction

Deep reinforcement learning algorithms are an impressive contribution of the deep learning era as they allow agents to learn different control tasks directly from interaction with an environment. However, a large number of training samples are necessary to optimally train the models. While learning from limited data or tasks, deep RL algorithms suffer from poor generalization performance when applying the learned policy to an unseen scenario. Raileanu and Fergus [2021] show that policy-value representation asymmetry is an underlying cause for poor generalization performance in shared actor-critic architectures. Value estimation in a particular state depends on instance-specific features; however, learning the policy only depends on task-specific features. Raileanu et al. shows that ignoring this asymmetry while learning a joint representation for both policy and value through a shared network limits the agent's capacity to learn a generalizable policy. Combining the two representations through a shared network guides the policy to be unnecessarily biased to the value features. Thus the learned policy overfits to the training instances and performs poorly in terms of generalization.

Previously proposed solutions to this asymmetry use two different networks for policy and value that capture distinct features [Cobbe et al., 2021][Raileanu and Fergus, 2021]. However, the use of two separate network creates a significant bottleneck regarding computation time. Also, the dependency of policy function on the value gradients requires extra precaution in designing the network. Nafi

Presented at Deep RL Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

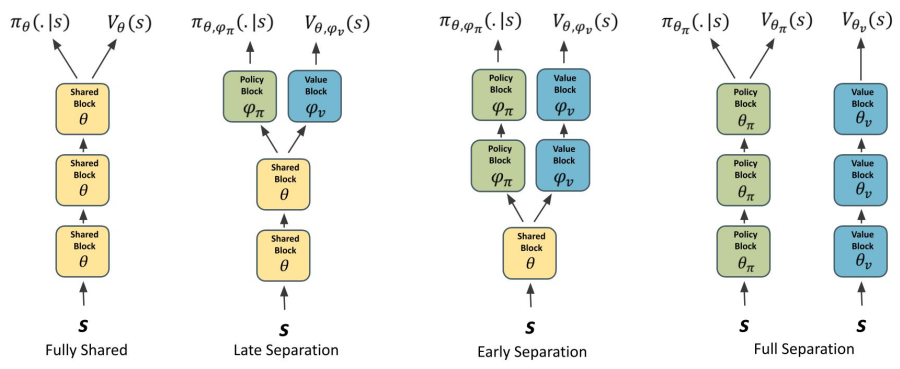

Figure 1: Architectures with different extent of decoupling for policy and value networks

et al. [2021] propose a workaround that acts as a compromise through partial separation of the policy and value networks and attains competitive performance compared to the fully separated counterpart while requiring less computational time. This architecture also eliminates the need for an additional value (or advantage) head to propagate the value gradient to the policy parameters. All of the abovementioned approaches were evaluated on the demanding generalization benchmark Procgen [Cobbe et al., 2020]. Procgen offers sixteen different game environments that provide access to the unlimited number of levels of a particular game with diverse backgrounds and game attributes such as agent position, shape and color of the game assets, enemy spawn time etc. The game level target remains the same throughout the levels, and the learned agent needs to perform well in the complete distribution of levels while learning from a limited number of levels.

We observe that while all these approaches offer a solution to the specific problem they identified, no solution is perfect for all sixteen environments considering both aspects - generalization performance and required computation time. Full separation of policy and value introduce very high computational overhead. In our experiments on a single GPU, for most of the environments, we observed 4 times higher running time for the fully separated approach, IDAAC [Raileanu and Fergus, 2021], compared to the fully shared baseline PPO [Cobbe et al., 2020] or partially separated approach APDAC [Nafi et al., 2021]. On the other hand, the partial separation achieves competitive performance in most environments, but fails in some cases.

Thus, the question remains open as to what approach should be used if we encounter a new environment. Unfortunately, there still does not exist a guideline for deciding how much decoupling is enough for policy and value networks in order to achieve a reasonable performance by looking at the environment properties, while keeping the computational burden low. In most cases, substantial compromise in generalization performance is not acceptable just because of the increased computational overhead. This also leads one to the question *"Can separating the network early (having most of the layers separated) improve the overall performance?"*

In this work, we investigate the effect of different degrees of decoupling on the agent's performance. Our analysis primarily leverages existing literature, and at the same time devises new purposeful insights. We evaluate four architectures with different degrees of separation: fully shared (no separation), early separation, late separation, and full separation. We attempt to delve into the learned representation for policy and value networks in all these settings to identify whether and how the separation helps. We conclude that full separation is a must when there is a clear distinction between the source of value and policy features. Otherwise, partial *late separation* suffices and possibly is the best option. However, too early separation may decrease the performance

2 Related Work

A lot of emphasis in deep reinforcement learning has lately been placed on an agent's ability to learn policies that are robust and generalizable Farebrother et al. [2018], Packer et al. [2018], Cobbe et al. [2019a] which has led to an emphasis on developing intelligent agents that avoid overfitting and can generalize well to unseen data Rajeswaran et al. [2017], Justesen et al. [2018], Grigsby and Qi [2020], Lyle et al. [2022]. Some of the methods that have been proposed include regularization techniques like dropout Igl et al. [2019], batch normalization Cobbe et al. [2019a], Hu et al. [2021], network randomization Lee et al. [2019] and data augmentation Cobbe et al. [2019b], Wang et al. [2020], Raileanu et al. [2020], Yarats et al. [2020], Zhang and Guo [2021]. A feature-swapping regularization technique to avoid observational overfitting is proposed in Bertoin and Rachelson [2022] whereas Igl et al. [2020], Lyle et al. [2022] use policy distillation to improve generalization. Zhang et al. [2020], Agarwal et al. [2021a] use bisimulation metrics to study similarity between states to learn task-relevant representations. Some studies suggest a link between interference and generalization in temporal difference learning and show that TD methods lead to under-generalizing parameters Bengio et al. [2020]. Mazoure et al. [2020] propose to predict the future states by maximizing the mutual information between its internal representation of successive time steps. Jia et al. [2022] introduce generalist-specialist training framework, while Paischer et al. [2022] propose the use of language models with history compression which enables memory to store abstractions of the observations to allow for generalization. One recent work identifies the value network as being much more prone to overfitting and propose a Delayed-Critic Policy Gradient (DCPG) method which trains the value function less frequently and with more training data compared to the policy Moon et al. [2022]. Nafi et al. [2022] propose hyperbolically discounted advantage-based policy learning for better generalization.

Another aspect of research focuses on the network architecture, such as the work of Raileanu and Fergus [2021] that uses decoupled policy and value networks to improve generalization and show that sharing policy and value functions leads to overfitting. In this context, while PPO Schulman et al. [2017] uses a fully shared actor-critic architecture, Nafi et al. [2021] propose the use of partially decoupled actor and critic networks that reduce the overall parameter count while performing comparably to the fully decoupled architecture. Cobbe et al. [2021] introduce a phase-wise training using two different networks and optimize the value function through an auxiliary phase. Several other works use decoupled architecture, however, the main objective of such works is to improve sample efficiency Andrychowicz et al. [2021] Yarats et al. [2021]. Recently, Ni et al. [2022] presents that recurrent neural network can be used with a decoupled architecture to perform better in POMDP settings which include generalization. However, there is hardly any work that attempts to measure the variation in performance due to different degrees of decoupling.

3 Methodology

To analyze the sensitivity to decoupling of policy and value, we create four different scenarios (1) there is no separation of the policy and value network other than the final policy and value heads (fully connected layers) at the end, (2) the network is separated early in the layers (3) the network is separated in the more downstream layers (4) solely two separate networks. Figure 1 presents the four architectures. We use the large IMPALA-CNN architecture [Espeholt et al., 2018] as the base network, as used by previous state-of-the-art works [Cobbe et al., 2021] [Raileanu and Fergus, 2021]. The Procgen benchmark [Cobbe et al., 2020] also used the same version of IMPALA-CNN architecture to present their results. This deeper IMPALA CNN architecture has three blocks, each having a configuration of Convolution Layer - Pooling Layer - Residual Block - Residual Block, where each residual block has two convolution layers. The whole architecture includes 15 convolutional layers in total [Espeholt et al., 2018]. In the rest of this section, we describe the details of these four architectures and their implementation using the IMPALA-CNN backbone. We also describe their loss functions and the optimization process.

3.1 Fully Shared

By *fully shared* or *no separation*, we denote the traditional architecture that shares the network for the actor (policy) and the critic (value). Specifically, we share the IMPALA-CNN base network (including all convolution layers) between policy and value, however, we only separate the final fully connected layers that represents the policy and value heads (see figure 1). This is similar to existing Proximal Policy Optimization (PPO) [Schulman et al., 2017] implementations e.g. [Cobbe et al., 2020] [Raileanu and Fergus, 2021]. Given the network parameter θ, we optimize the standard objective function used in policy gradient approaches:

$$J_{NS}(\theta) = J_{\pi}(\theta) - \alpha_v L_V(\theta) + \alpha_s S_{\pi}(\theta) \tag{1}$$

where $J_\pi(\theta)$ is the policy gradient objective, $L_V(\theta)$ is the value loss, $S_\pi(\theta, \phi_\pi)$ is the entropy bonus for exploration, and α_v and α_s are the corresponding coefficients. PPO builds upon the Trust Region Policy Optimization (TRPO) Schulman et al. [2015] method, which maximizes a surrogate objective function defined as:

$$J_\pi(\theta) = \hat{\mathbb{E}}_t\Big[r_t(\theta)\hat{A}_t\Big] \tag{2}$$

where $r_t(\theta) = \frac{\pi_{(\theta)}(a_t|s_t)}{\pi_{(\theta)_{old}}(a_t|s_t)}$ is the probability ratio between the new policy and the old policy, and $\hat{A}_t$ refers to the advantage estimate at timestep t. To avoid excessively large policy updates, PPO clips the value of $r_t(\theta)$ between the intervals of $[1-\epsilon, 1+\epsilon]$ and takes the minimum between the original value of $r_t(\theta)$ and the clipped one. Thus, the final clipped surrogate objective function is as follows:

$$J_\pi(\theta) = \hat{\mathbb{E}}_t\Big[min\big(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big] \tag{3}$$

For the fully shared architecture, we optimize the same clipped surrogate objective as shown in Equation 1.

3.2 Early Separation

To implement early separation, we share the first block of the IMPALA-CNN architecture and separate the next two for policy and value. Thus, only 5 convolutional layers are shared. Consider that the part of the network shared between policy and value is parameterized by θ, the separated policy subnetwork is parameterized by ϕ_π, and the separated value subnetwork is parameterized by ϕ_v. Then, we optimize the objective jointly for all parameters, similar to the PPO loss (eq 1):

$$J_{ES}(\theta, \phi_\pi, \phi_v) = J_\pi(\theta, \phi_\pi) - \alpha_v L_V(\theta, \phi_v) + \alpha_s S_\pi(\theta, \phi_\pi) \tag{4}$$

where $J_\pi(\theta, \phi_\pi)$ is the policy gradient objective, $L_V(\theta, \phi_v)$ is the value loss and $S_\pi(\theta, \phi_\pi)$ is the entropy bonus.

3.3 Late Separation

In this variant, we share the first two blocks of the IMPALA-CNN architecture and use separate third block for each of the policy and value network components. As a result, the first 10 convolutional layers are shared while there are two different sets of 5 convolutional layers for each of the policy and value subnetworks. If the shared network part is parameterized by θ, the separated policy network and value network are parameterized by ϕ_π and ϕ_v respectively, we optimize the objective jointly, similar to the PPO loss (eq 1):

$$J_{LS}(\theta, \phi_\pi, \phi_v) = J_\pi(\theta, \phi_\pi) - \alpha_v L_V(\theta, \phi_v) + \alpha_s S_\pi(\theta, \phi_\pi) \tag{5}$$

where $J_\pi(\theta, \phi_\pi)$ is the policy gradient objective, $L_V(\theta, \phi_v)$ is the value loss and $S_\pi(\theta, \phi_\pi)$ is the entropy bonus. This is similar to the partial separation proposed in [Nafi et al., 2021]. The difference between early separation and late separation is that the number of shared parameters represented by θ is higher in late separation.

3.4 Full Separation

As the name suggests, we use two fully separate network for policy and value. This is similar to the one proposed in [Raileanu and Fergus, 2021] that fully disentangle the policy and value representations. We also keep the extra advantage head in the policy network as without any sort of value or advantage gradient, the policy network remains isolated, resulting in the policy optimization process failing completely [Cobbe et al., 2021]. Since there are two different networks, the optimization takes place in two phases with the policy network optimized at first, followed by the value network. The policy network parameterized by θ_π is optimized for:

$$J_{FS}(\theta_\pi) = J_\pi(\theta_\pi) - \alpha_A L_{A_\pi}(\theta_\pi) + \alpha_s S_\pi(\theta_\pi) \tag{6}$$

Here $L_{A_\pi}(\theta_\pi)$ is the advantage loss coming from the additional advantage head used to support the policy network [Raileanu and Fergus, 2021]. On the other hand, the value network, parameterized by ϕ_v, optimizes the value loss defined as follows:

$$L_v(\theta_v) = \hat{\mathbb{E}}_t[(V_{\theta_v}(S_t) - \hat{V}_t^{targ})^2]$$

Where $\hat{V}_t^{targ}$ is the target value function.

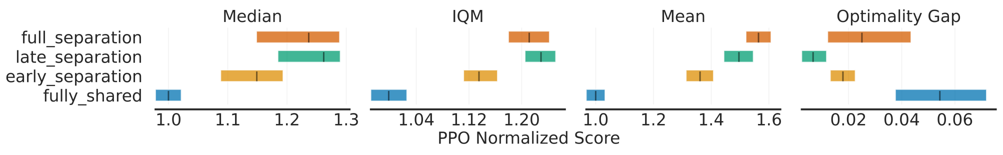

Figure 2: PPO normalized score for all four variants: fully shared, early separation, late separation, and full separation across all the 16 environments in Procgen benchmark

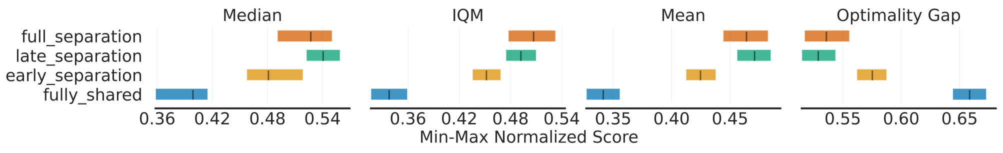

Figure 3: Min-Max normalized score for all four variants: fully shared, early separation, late separation, and full separation across all the 16 environments in Procgen benchmark

4 Results and Discussions

We conduct our experiments against all the 16 game environments available in the Procgen benchmark [Cobbe et al., 2020].[1] The highly diverse environments provide us the opportunity to analyze and draw conclusions regarding when to use a certain network architecture. We use the same protocol for training and testing as introduced in Cobbe et al. [2020]. We employ the easy distribution mode and train all the agents on only 200 levels of the games while testing on the full distribution of levels. The term full distribution refers to the configuration that each episode in the testing phase can be any level selected from a infinite set of procedurally generated levels. Thus the learned policy needs to perform better in the unseen scenarios, not encountered during training.

4.1 Generalization Across All Environments

To analyze the overall performance, we first report the results combined across all 16 environments. The general evaluation criteria in reinforcement learning is the average test scores (returns) achieved on evaluation episodes while training the model on 25M timesteps. To address statistical uncertainty, we consider more critical metrics such as Interquartile Mean (IQM) and optimality gap (OG), in addition to mean and median scores across all runs, as introduced by Agarwal et al. [2021b]. Further, we present performance profiles that takes into account the variability in performance across tasks and runs. A detailed description of these metrics is available in the supplementary materials.

Figure 2 shows the PPO-normalized scores across all the 16 environments. As described previously in Section 3.1, *fully shared* approach refers to the PPO algorithm. Thus, figure 2 can be considered as a performance representation of other network architectures relative to the fully shared version wherein the score of the *fully shared* version is 1 in the scale. It is evident from figure 2 that *late separation* outperforms all other approaches in terms of the IQM value across all environments while sharing a significant portion of the network. While the median value of *late separation* is also higher than any other approaches, the mean value is slightly less compared to the *full separation* but still better than *early separation*. Further, the optimality gap of *late separation* is the lowest among all other approaches.

Figure 3 shows the min-max normalized score across all the 16 environments. Min-max normalized scores can be viewed as the performance compared to the minimum and maximum scores achievable in an environment, values of which are provided in Cobbe et al. [2020]. We observe that the mean score of *late separation* performs best, however, the IQM value is slightly less than *full separation*. Both the median and optimality gap are better in case of *late separation*.

From the presented results, we observe that *early separation* while having a large portion of the network separated for the policy and value still performs worse than the *late separation*. However, a

[1]https://github.com/nasiknafi/sensitivity-to-decoupling

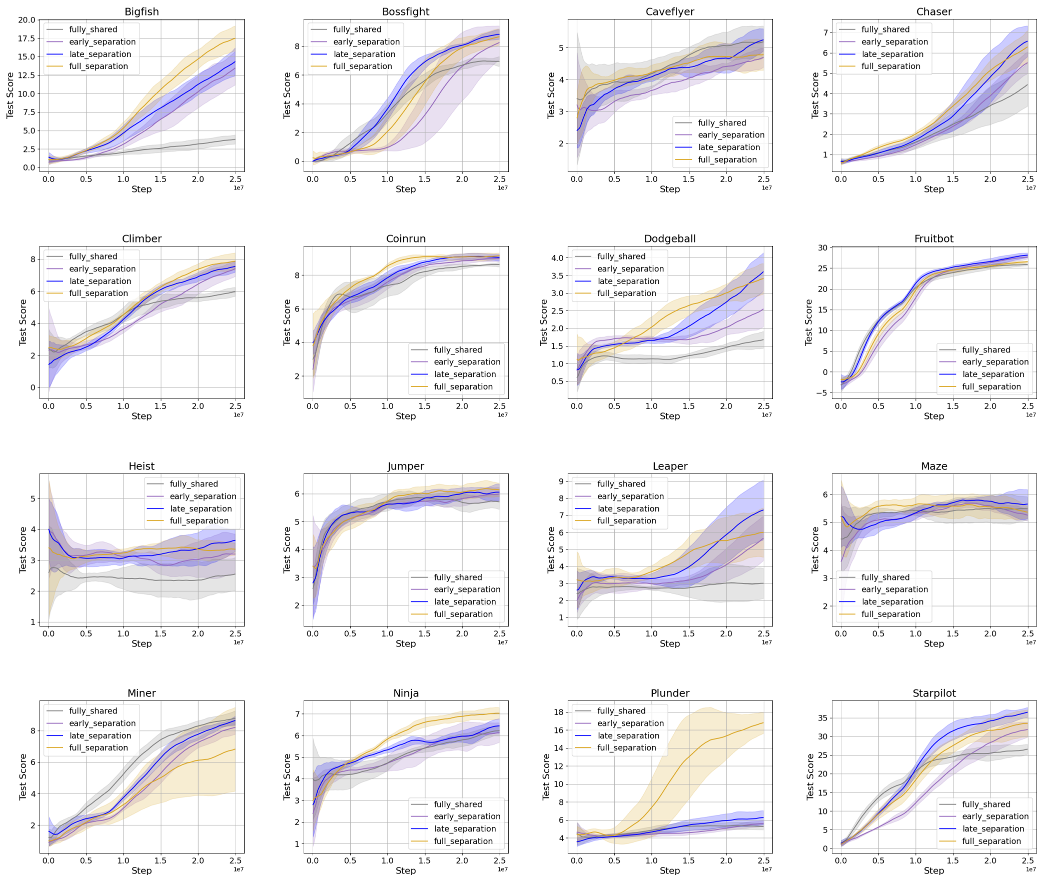
Figure 4: Test performance for fully shared, early separated, lately separated, and fully separated architecture across all 16 environments from the Procgen benchmark. Mean and standard deviation are calculated over 5 trials, each with a different seed.

Table 1: Comparison of computational time and required convolutional layers

Methods	No of Conv. Layer	GPU Hours
fully shared	15	3-4
late separation	20	3-5
early separation	25	3-5
full separation	30	5-10

naive assumption could lead one to believe that early separation architecture would work best as this should capture the policy-value representation asymmetry better through its larger separated part. We hypothesize that this deviation is due to the combined adversarial effect of the higher isolation of the policy network and the lack of separate policy and value optimization. The rationale behind this argument refers to the addition of advantage head (or value head) to the policy network and separate optimization of the policy and value networks to alleviate performance degradation that occurs with naive separation Raileanu and Fergus [2021] Cobbe et al. [2021].

4.2 Performance on Individual Environments

In addition to the overall performance, we look at the empirical results for all individual environments to evaluate how different degrees of separation affects them. Figure 4 presents results across the entire Procgen benchmark. The solid line refers to the running mean of rewards while the shaded regions refer to the variance across trials. We observe that, in all the environments except *Plunder*,

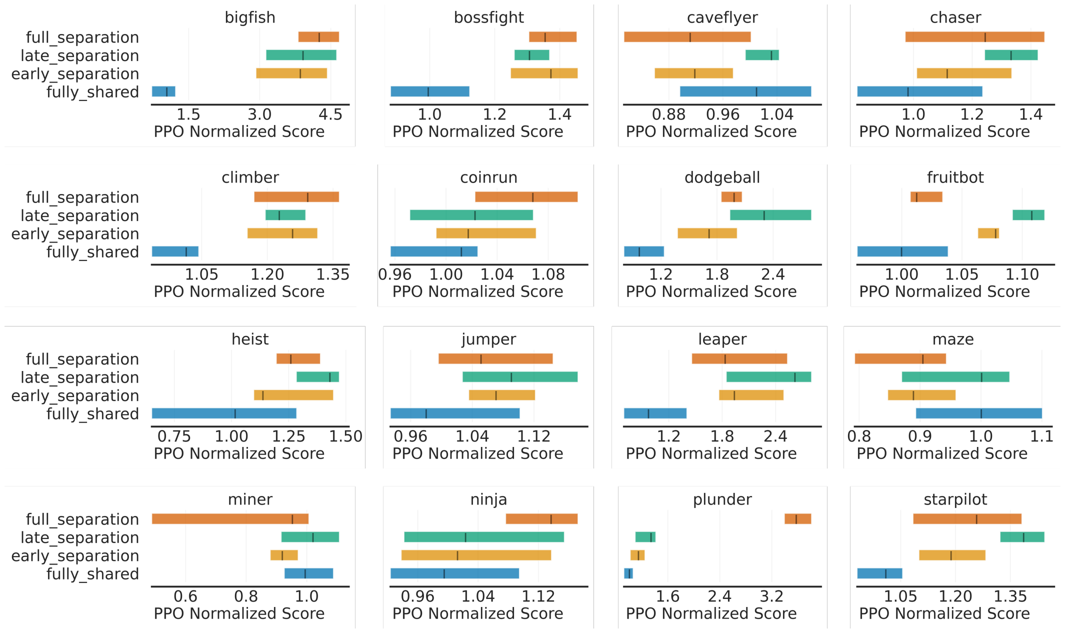

Figure 5: Interquartile Mean (IQM) for PPO (Fully Shared) normalized test rewards for all the 16 environment. Mean and standard deviation are calculated over 5 trials, each with a different seed.

late separation achieves competitive scores, even better in some cases, against the model that uses *full separation* architecture. *early separation* performs competitively with *late separation* in a few environments, however this performance is not consistent. The *fully shared* architecture one only performs competitively in the *caveflyer* and *miner* environments.

Figure 5 presents the PPO-normalized IQM values for each of the environments to assess the performance difference in a statistically significant way. We observe similar behavior as in figure 4. The drastic failure of the fully shared or any of the partially separated variants in *plunder* environment is also visible from the IQM plots. Comparison of other metrics such as Mean, Median and Optimality Gap for each environment is available in the supplementary materials. Table 1 shows a comparison of the time complexity for all the four variants used in our experiments. As partially separated models requires significantly less time than the fully separated approach and *late separation* can achieve competitive performance in most cases, we recommended to use *late separation* in general and specially when computational complexity is crucial.

We now examine the issue of complete failure of all architectures except full separation in the *Plunder* environment. Our investigation reveals that unlike other environments, *Plunder* includes an on-screen timer that slowly counts down (see the top green bars in the left figures in Figure 6). When the timer runs out, the episode ends at that instant. Thus this on-screen timer acts as a life bar for the agents. The policy needs to learn to avoid hitting friendly ships and destroying the enemy pirate ships by firing cannonballs. A target in the bottom left corner of the screen shows the color of the targeted enemy ships. However, the life bar is an important source of value estimation of the state. Figure 6 presents examples of the learned policy and value representation highlighted through Grad-CAM Selvaraju et al. [2017] for both *late separation* and *full separation*. It is evident that the value representation in the fully separated one pays attention at the top-left corner where the life bar ends while the value network in case of late (partial) separation fails to do so. On the other hand, the policy representations for fully separate network keeps track of the end of the life bar while also paying attention to the enemy ships. However, the policy representations in case of late (partial) separation focuses at the complete life bar. We hypothesize that this happens due to the overarching effect of the value function. The policy network puts significant importance on the value source and fails to distinguish between the policy and value representation. Thus, we conclude that when explicit source of value estimation is present in the input observation, any one should consider learning representation for policy and value through two fully separate network irrespective of the computational time overhead to gain generalization improvements.

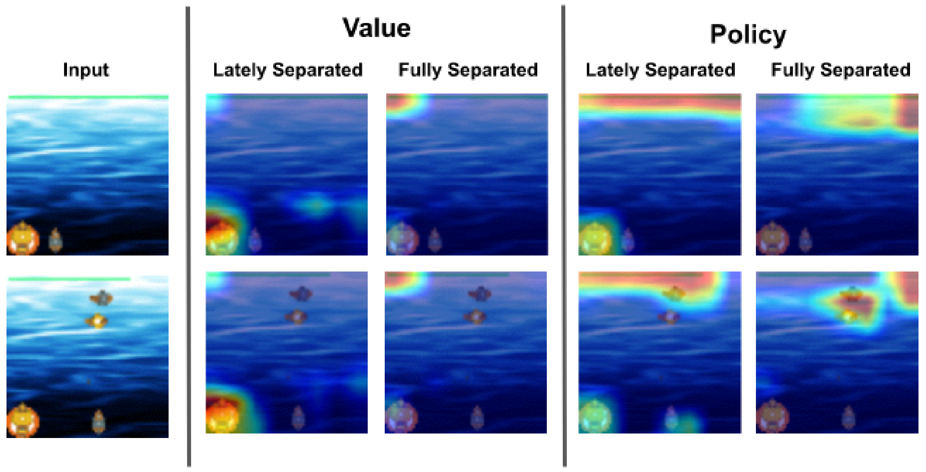

Figure 6: Learned representation of the policy and value using lately separated and fully separated architectures for the Plunder environment. The red marked regions correspond to the higher value of gradients while blue regions correspond to the lower value of gradients. The learned policy representation using lately (partially) separated approach get biased with the value function and primarily looks at the whole life bar as opposed to the end mark focused by the policy representation learned by the fully separated approach.

5 Conclusion

In this work, we attempt to infer meaningful insights from the existing works through carefully designed ablation studies regarding partial and full decoupling of policy and value networks for generalization in reinforcement learning. We present comparative analysis of the models that use different level of decoupling/separation for the policy and value function in procedurally generated environments. Our work clearly identifies when to leverage two fully separate networks even though it might entail increased computational complexity. Our contribution thus focuses on and recommends practical criteria that will help deep reinforcement learning practitioners in deciding the type of network and extent of decoupling to deploy when encountering a new environment to achieve better generalization.

References

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

Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

Nasik Muhammad Nafi, Creighton Glasscock, and William Hsu. Attention-based partial decoupling of policy and value for generalization in reinforcement learning, 2021.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings* *of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 09–15 Jun 2019a. URL `https://proceedings.mlr.press/v97/cobbe19a.html`.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. *arXiv preprint arXiv:1703.02660*, 2017.

Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv preprint arXiv:1806.10729*, 2018.

Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *CoRR*, abs/2010.06740, 2020. URL `https://arxiv.org/abs/2010.06740`.

Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14560–14581. PMLR, 17–23 Jul 2022.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. *arXiv preprint arXiv:1910.12911*, 2019.

Tianyang Hu, Wenjia Wang, Cong Lin, and Guang Cheng. Regularization matters: A nonparametric perspective on overparametrized neural network. In *International Conference on Artificial Intelligence and Statistics*, pages 829–837. PMLR, 2021.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2019.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019b.

Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *Advances in Neural Information Processing Systems*, 33: 7968–7978, 2020.

Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2020.

Hanping Zhang and Yuhong Guo. Generalization of reinforcement learning with policy-aware adversarial data augmentation. *arXiv preprint arXiv:2106.15587*, 2021.

David Bertoin and Emmanuel Rachelson. Local feature swapping for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2022.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

Rishabh Agarwal, Marlos C Machado, Pablo Samuel Castro, and Marc G Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *arXiv preprint arXiv:2101.05265*, 2021a.

Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pages 767–777. PMLR, 2020.

Bogdan Mazoure, Remi Tachet des Combes, Thang Long Doan, Philip Bachman, and R Devon Hjelm. Deep reinforcement and infomax learning. *Advances in Neural Information Processing Systems*, 33:3686–3698, 2020.

Zhiwei Jia, Xuanlin Li, Zhan Ling, Shuang Liu, Yiran Wu, and Hao Su. Improving policy optimization with generalist-specialist learning. In *International Conference on Machine Learning*, pages 10104–10119. PMLR, 2022.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17156–17185. PMLR, 17–23 Jul 2022.

Seungyong Moon, JunYeong Lee, and Hyun Oh Song. Rethinking value function learning for generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 2022.

Nasik Muhammad Nafi, Raja Farrukh Ali, and William Hsu. Hyperbolically discounted advantage estimation for generalization in reinforcement learning. In *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=nIAxjsniDzg`.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021.

Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. In *International Conference on Machine Learning*, pages 16691–16723. PMLR, 2022.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain 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.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021b.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

Roberta Raileanu. Pytorch implementation of daac and idaac. `https://github.com/rraileanu/idaac`, 2021.

Nasik Muhammad Nafi. Pytorch implementation of apdac. `https://github.com/nnafi/apdac`, 2021.

A Implementation and Hyperparameters

For PPO Schulman et al. [2017] as a representative algorithm for fully shared networks, we use implementation released by [Kostrikov, 2018]. For fully separated networks, we use the code released [Raileanu, 2021] as part of the IDAAC paper [Raileanu and Fergus, 2021]. For early and late separation networks, we use APDAC [Nafi et al., 2021] and modify the code released with the paper [Nafi, 2021]. Each environment for a particular algorithm was run using five different seed values. The software used included Python (3.7.x) and PyTorch (1.7.1), with CUDA (10.2). Our computing infrastructure involved using a High Performance Cluster with GPUs including Nvidia RTX 2080s. Each experiment was run using one GPU, and took between 3 hours to 10 hours, depending on the degree of network separation (see Table 1 of main paper). We provide the hyperparameters for our algorithms in Table 2, as well as the configurations that were used in our environments (Table 3) for reproducibility. These values are used in all our experiments unless specified otherwise.

Table 2: Hyperparameters used for all policy gradient methods

hyperparameters	**values**
timesteps per rollout	256
epochs per rollout	3
minibatches per epoch	8
entropy bonus	0.01
clip range	0.2
reward normalization	yes
learning rate	5e-4
environments per worker	64
total timesteps	25 million
Optimizer	Adam
GAE lambda λ	0.95

Table 3: Procgen environment configurations

configuration	**values**
distribution mode	Easy
num (train)	200
start level	0
paint vel info	False
center agent	True
use sequential levels	False
use generated assets	False
use backgrounds	True
restrict themes	False
use monochrome assets	False

B Train performance

Figure 7 shows the train performance of the four network separation methods on all Procgen environments.

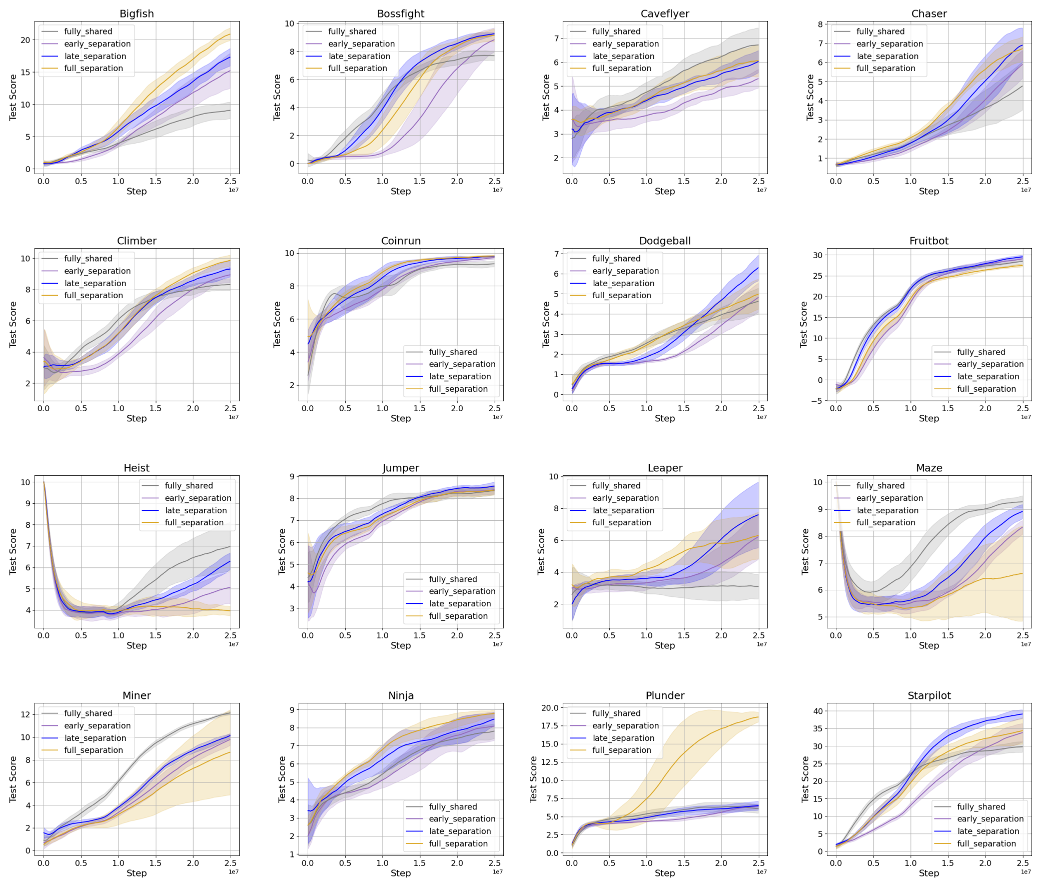

Figure 7: Train performance of the four architecture: fully shared, early separation, late separation, and full separation for all the 16 environment of Procgen benchmark. Mean and standard deviation are calculated over 5 trials, each with a different seed.