Direct Advantage Estimation

Anonymous Author(s)
Affiliation
Address
email

Abstract

The predominant approach in reinforcement learning is to assign credit to actions based on the expected return. However, we show that the return may depend on the policy in a way which could lead to excessive variance in value estimation and slow down learning. Instead, we show that the advantage function can be interpreted as causal effects and shares similar properties with causal representations. Based on this insight, we propose Direct Advantage Estimation (DAE), a novel method that can model the advantage function and estimate it directly from on-policy data while simultaneously minimizing the variance of the return without requiring the (action-)value function. We also relate our method to Temporal Difference methods by showing how value functions can be seamlessly integrated into DAE. The proposed method is easy to implement and can be readily adapted by modern actor-critic methods. We evaluate DAE empirically on three discrete control domains and show that it can outperform generalized advantage estimation (GAE), a strong baseline for advantage estimation, on a majority of the environments when applied to policy optimization.

1 Introduction

Reinforcement learning (RL) methods aim to maximize cumulative rewards in sequential decision making problems [Sutton and Barto, 2018]. Through interactions with the environment, agents learn to identify which actions lead to the highest return. One major difficulty of this problem is that, typically, a vast amount of decision makings are involved in determining the rewards, making it complex to identify which decisions are crucial to the outcomes. This is also known as the credit assignment problem [Minsky, 1961]. A straightforward approach is to assign credit according to the expected return, which is aligned with the RL objective[Sutton and Barto, 2018]. However, the expected return accumulates all the future rewards and does not directly reflect the effect of an action. Consider, for example, the environment illustrated in Figure 1. Since the transitions at each state are independent of the actions chosen, the *effect* of any action should be immediate. The expected return conditioned on any action, however, depends on the policy of all subsequent states. This property could be undesirable because the expected return would vary significantly whenever the policy changes during training, making it a difficult target to learn.

A natural question to ask is then, how should we define the *effect* of an action? In this work, we take inspiration from the causality literature, where the notion of causal effects is defined. We show that the advantage function can be interpreted as the causal effect of an action on the expected return. Additionally, we show that it shares similar properties with causal representations, which are expected to improve generalization and transfer learning [Schölkopf et al., 2021].

We prove that the advantage function minimizes the variance of the return under a normalization constraint, and propose **Direct Advantage Estimation** (DAE), a method that can model and estimate the advantage function directly from sampled trajectories unlike previous methods that rely on learning either the value function or the Q-function beforehand. Furthermore, we show that DAE

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

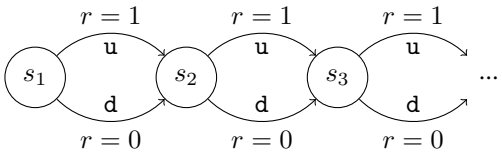

Figure 1: An example environment where the effect of an action on the return is only immediate. The agent starts in state s_1 and, at each step t, chooses either the action u (up) or d (down) with immediate rewards of 1 or 0, respectively, before transitioning into the next state s_{t+1}.

can be combined with a value function approximator and be updated in a way similar to Temporal Difference methods [Sutton, 1988]. We then illustrate how DAE can be integrated into modern deep actor-critic methods using the example of Proximal Policy Optimization (PPO) [Schulman et al., 2017]. Finally, we test our method empirically on three discrete domains including (1) a synthetic environment, (2) the MinAtar suite [Young and Tian, 2019] and (3) the Arcade Learning Environment [Bellemare et al., 2013], and demonstrate that DAE outperforms Generalized Advantage Estimation (GAE) [Schulman et al., 2015b] on most of them.

The main contributions of this paper are

- We analyze the properties of the advantage function and show how they are linked to causality.
- We develop DAE, a novel on-policy method for estimating the advantage function directly, and compare it empirically with GAE using PPO.

2 Background

We consider a discounted Markov Decision Process $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ with finite state space $\mathcal{S}$, finite action space $\mathcal{A}$, transition probability $P(s'|s,a)$, expected reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and discount factor $\gamma \in [0, 1)$. A policy is a function $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $\pi(a|s)$ denoting the probability of action a given state s. A trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$ is said to be sampled from a policy π if $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$. We denote the return of a trajectory by $G(\tau) = \sum_{t\geq 0} \gamma^t r_t$, where $r_t = r(s_t, a_t)$. The goal of an RL agent is to find a policy π^* that maximizes the expected return, i.e., $\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}[G(\tau)]$. Given a policy π, we define the value function by $V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[G(\tau)|s_0{=}s]$, the action-value function by $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[G(\tau)|s_0{=}s, a_0{=}a]$, and the advantage function by $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ [Sutton and Barto, 2018, Baird, 1993]. For simplicity, we shall omit the superscript. These functions quantify how good a certain state (state-action pair) is, which can be used to improve the policy. In practice, however, they are usually unknown to the agent, and have to be learned from its interactions with the environment. Below, we briefly summarize how they are typically learned.

Monte Carlo (MC) method. By definition, an unbiased way to estimate $V(s)$ is by sampling trajectories starting from state s and averaging the sample returns $V(s) \approx \frac{1}{n}\sum_{i=1}^{n} G(\tau_i)$. While being conceptually simple and unbiased, MC methods are known to suffer from high variance, which can cause slow learning.

Temporal Difference (TD) learning. A different approach based on bootstrapping is proposed by Sutton [1988]. In this approach, we begin with an initial guess of the value function V and improve our estimates iteratively by sampling transitions (s, a, r, s') and updating our current estimates to match the bootstrapping targets via $V(s) \leftarrow V(s) + \alpha(r + V(s') - V(s))$. This is also known as the TD(0) method. Unlike MC methods, TD methods have lower variance at the cost of being biased due to their use of bootstrapping targets. Methods based on bootstrapping have been shown to converge faster than MC methods empirically and have been widely adapted in recent deep RL methods [Mnih et al., 2015, 2016, Schulman et al., 2017].

One disadvantage of the TD(0) method is that it only propagates the reward one step at a time, which can be slow when the rewards are sparse. Instead, one can utilize the rewards that are up to n-steps ahead and update value estimates by $r_t + \gamma r_{t+1} + ... + \gamma^n V(s_{t+n})$. This allows the reward

information to propagate faster to relevant state-action pairs. Additionally, this view unifies both methods in the sense that TD(0) methods correspond to the case when $n = 1$ while $n \to \infty$ recovers MC methods. Choosing the parameter n can be seen as a bias and variance trade-off, where lower n relies more on bootstrapping which could be biased while higher n relies more on the sampled trajectory which is less biased but could lead to larger variance. In TD(λ), these n-step estimations are further combined to produce a more robust estimate by averaging them exponentially with λ^n [Sutton, 1988, Watkins, 1989].

TD errors and the advantage function. One way to estimate $A(s, a)$ is to sample a transition (s, a, r, s') and compute $A(s, a) = Q(s, a) - V(s) \approx r + \gamma V(s') - V(s)$. The term $r + \gamma V(s') - V(s)$ is known as TD Error and was introduced in the TD Learning algorithm to update the value estimates. This method can also be combined with TD(λ) to produce more accurate estimates of the advantage function [Schulman et al., 2015b].

Policy optimization. One important application of the advantage function is in policy optimization. Williams [1992], Sutton et al. [2000] showed that, for a parameterized policy π_θ, the gradient of the RL objective is given by

$$\nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[G(\tau)] = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \left(Q(s, a) - b(s)\right) \nabla_\theta \pi_\theta(a|s), \tag{1}$$

where $d^\pi(s) = \sum_{t \geq 0} \gamma^t p(s_t{=}s)$ and $b(s)$ is an arbitrary function of the state known as the baseline function. The baseline function can be seen as a control variate, which can be used to reduce the variance of the gradient estimator when chosen carefully [Greensmith et al., 2004]. One common choice of $b(s)$ is the value function V, which results in the advantage function.

In a different approach, Kakade and Langford [2002] showed that the difference between the expected return of two policies is directly related to the advantage function, which paved the way for some of the most popular policy optimization methods [Schulman et al., 2015a, 2017].

3 Revisiting the advantage function

We begin by examining the motivating example in Figure 1. In this example, it is clear that the optimal decision depends solely on the immediate reward at each state, since the transitions are independent of the actions. However, if we calculate the Q-function, we get

$$Q^\pi(s_t, a_t) = \begin{cases} 1 + \sum_{t' > t} \gamma^{t'-t} \pi(\mathtt{u}|s_{t'}), & a_t = \mathtt{u} \\ 0 + \sum_{t' > t} \gamma^{t'-t} \pi(\mathtt{u}|s_{t'}), & a_t = \mathtt{d} \end{cases} \tag{2}$$

which shows that the Q-values at each state depend on the policy evaluated at all subsequent states despite the actions having only immediate effects on the expected return. This property can be undesirable as the policy is typically being optimized and varied, which could lead to variations in the Q-values and make previous estimates unreliable. This is also known as the problem of *distribution shift*, where the data generating distribution differs from the distribution we are making predictions on. Similar problems have also been studied in offline RL, where distribution shift can cause Q-function estimates to be unreliable Levine et al. [2020], Kumar et al. [2020], Fujimoto et al. [2019]. One source of this problem is that the expected return by nature associates an action with all its subsequent rewards, whether they are causally related or not. Alternatively, we would like to focus on the *effect* of an action. But how should we define it?

Questions like this are central to the field of causality and have been studied extensively [Holland, 1986, Pearl, 2009]. We draw our inspiration from Rubin [1974] in which the author developed a framework, now known as the Neyman-Rubin model, to estimate causal effects of treatments. It defines the *causal effect* of a treatment to be the difference between (1) the outcome of a trial resulting from one treatment, and (2) what would have happened if we had chosen another treatment. The work focused mainly on the problem of counterfactuals, which we shall not delve into. Instead, we focus simply on its definition of causal effects. In RL, we concern ourselves with the effect of each action on the expected return. However, the action space usually consists of more than two actions, so it is ambiguous as to which action we should compare to. One way to resolve this is to compare

each action with what would have happened normally [Halpern and Hitchcock, 2015]. This way, we can define the causal effect of an action a on some quantity X at state s by

$$\mathbb{E}\left[X|s,a\right] - \mathbb{E}\left[X|s\right]. \tag{3}$$

If we choose X to be the return, then

$$\mathbb{E}\left[G(\tau)|s_t = s, a_t = a\right] - \mathbb{E}\left[G(\tau)|s_t = s\right] = Q(s,a) - V(s) = A(s,a) \tag{4}$$

is precisely the advantage function, suggesting that it can be interpreted as the causal effect on the expected return. If we calculate the advantage function for the example environment in Figure 1, we get

$$A^{\pi}(s,a) = \begin{cases} 1 - \pi(\mathtt{u}|s), & a = \mathtt{u} \\ 0 - \pi(\mathtt{u}|s), & a = \mathtt{d} \end{cases} \tag{5}$$

which now depends solely on the policy evaluated at the state s. This is because, in this example, the future state distribution is independent of the actions, so the conditional expectation cancels out, leaving only the immediate term. More generally, we only need the expected return for the future state to be independent of the current action.

Proposition 1. Given $t' > t$ such that $\mathbb{E}\left[V(s_{t'})|s_t, a_t\right] = \mathbb{E}\left[V(s_{t'})|s_t\right]$, we have

$$A(s_t, a_t) = \mathbb{E}\left[\sum_{k=t}^{t'-1} \gamma^{k-t} r_k \middle| s_t, a_t\right] - \mathbb{E}\left[\sum_{k=t}^{t'-1} \gamma^{k-t} r_k \middle| s_t\right] \tag{6}$$

See Appendix A for proofs. This proposition shows that the advantage function is *local*, in the sense that the advantage of an action is only dependent on the policy of subsequent states to the point where the action stops being relevant to future rewards. This property is strikingly similar to the Sparse Mechanism Shift hypothesis [Schölkopf et al., 2021] which states that small distribution changes should only lead to sparse or local changes if the representation is causal/disentangled.

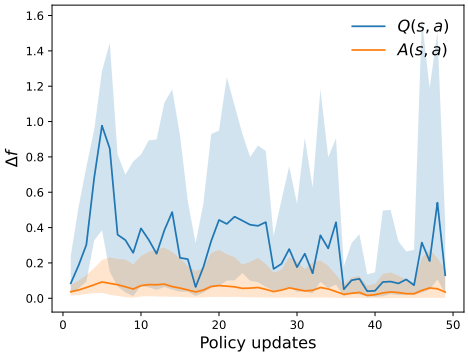

Figure 2: $\Delta f = |f^{\pi_i} - f^{\pi_{i-1}}|$ estimated using MC simulations between each policy update computed with 3072 state-action pairs sampled throughout training. Lines and shadings represent the median and the interquartile range over all sampled state-action pairs. See Appendix C for details.

The condition of Proposition 1 may be too strict to hold in most cases. As a relaxation, we hypothesize that, for a large class of problems of interest, the advantage function is more stable under policy variations compared to the Q-function, making it an easier target to learn. This may partially explain performance improvements reported by Baird [1993], Wang et al. [2016b]. In Figure 2, we verify this empirically using the Breakout environment from the MinAtar suite [Young and Tian, 2019] by tracking the variations of Q and A throughout the training of a PPO agent. We can see that the advantage function remains more stable over the course of training compared to the Q-function. From a more theoretical point of view, this softer condition is satisfied when $p(s_{t'}|s_t, a_t) \approx p(s_{t'}|s_t)$ for some $t' > t$. This can happen when, for example, the Markov chain induced by the policy approaches a stationary distribution; as another example, if the environment contains certain bottleneck states (e.g., doorways), that are traversed with high probability, then the state distribution after passing those states would depend weakly on the actions before passing them. On the other hand, if the environment is "tree-like", where actions lead to completely different branches, then the advantage function would also suffer from instability.

The question we are now interested in is how to learn the advantage function. In previous work, estimating the advantage function typically relied on using TD errors. This method also faces the problem of relying on an estimate of the value function that suffers from strong policy dependency. Instead, we seek methods which could directly model the advantage function and estimate it from

data. Since both Q and V can be learned with bootstrapping via TD learning, one might wonder if there is an equivalent for the advantage function. Here we give a counterexample showing that the advantage function cannot be learned in the same way.

Consider the environment in Figure 3. In this example, the undiscounted advantage function for s is also equal to Equation 5, while the advantages for both $s_{\mathtt{u}}$ and $s_{\mathtt{d}}$ are zero because there is only a single action in their action spaces. Since the immediate rewards and the advantages for the next states are all zero for the state s, we conclude that it is not possible to learn the advantages of s based solely on the immediate reward and the advantages of the next state as in TD learning. An intuitive explanation for this failure is that, unlike the value function, which contains information about all future rewards, the advantage function only models the causal effect of each action; therefore, if the next action is not causal to a distant reward, then the information about that reward is lost.

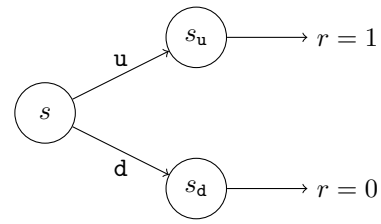

Figure 3: A synthetic environment demonstrating the impossibility of learning A from bootstrapping. An agent begins in state s with actions u and d which transition the agent into states $s_{\mathtt{u}}$ and $s_{\mathtt{d}}$, respectively, with no immediate reward. In both $s_{\mathtt{u}}$ and $s_{\mathtt{d}}$, there is only one action with immediate rewards 1 and 0, respectively.

4 Direct Advantage Estimation

In this section, we present the main contribution of this paper. We develop a simple method to model and estimate the advantage function directly from sampled trajectories.

Before we begin, we first observe that the advantage function satisfies the equality $\sum_{a\in\mathcal{A}} \pi(a|s)A^\pi(s,a) = 0$. Functions with this property, as it turns out, are useful for the development of our method, so we provide a formal definition.

Definition 1. Given a policy π, we say a function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is *π-centered* if for all states $s \in \mathcal{S}$

$$\sum_{a\in\mathcal{A}} \pi(a|s)f(s,a) = 0. \tag{7}$$

As discussed in Section 2, MC and n-step methods often suffer from excessive variance due to their sampling nature. It is therefore desirable to find unbiased low-variance estimates of the return. One interesting property of π-centered functions is that, if we transform the reward by $r'_t = r_t - \hat{A}(s_t, a_t)$ with any π-centered function $\hat{A}$, then the expected return remains unchanged, that is $\mathbb{E}\left[G(\tau)\right] = \mathbb{E}\left[G'(\tau)\right]$, where $G'(\tau) = \sum_{t\geq 0} \gamma^t r'_t$. This can be seen as a form of reward shaping [Ng et al., 1999], where the expected return is now invariant to the choice of $\hat{A}$ for a fixed policy. Now, consider the variance of the transformed return $G'(\tau)$: we have $\mathbf{Var}\left[G'(\tau)\right] = \mathbb{E}\left[G'(\tau)^2\right] - \mathbb{E}\left[G(\tau)\right]^2$. In other words, minimizing its variance with respect to $\hat{A}$ is equivalent to minimizing the following quantity,

$$\mathbb{E}\left[\left(\sum_{t=0}^{\infty} \gamma^t \left(r_t - \hat{A}(s_t, a_t)\right)\right)^2\right]. \tag{8}$$

Minimizing the variance of the return this way has the additional benefit of simultaneously obtaining the advantage function without having to learn the (action-)value function first, as we show in our main theorem.

Theorem 1. Given a policy π and time step $t \geq 0$, we say $(s,a) \in \mathcal{S} \times \mathcal{A}$ or $s \in \mathcal{S}$ is reachable within t if there exists $0 \leq t' \leq t$ such that $p(s_{t'}{=}s, a_{t'}{=}a) > 0$ or $p(s_{t'}{=}s) > 0$, respectively. Let F_π be the set of all π-centered functions, $\hat{A}_t = \hat{A}(s_t, a_t)$, and

$$\hat{A}^* = \operatorname*{arg\,min}_{\hat{A}\in F_\pi} \mathbb{E}\left[\left(G(\tau) - \sum_{t'=0}^{t} \gamma^{t'} \hat{A}_{t'}\right)^2\right],$$

then for all (s,a) that are reachable within t, we have $\hat{A}^*(s,a) = A^\pi(s,a)$.

Note that the first term inside the expectation is equal to the return, while the summation in the second term is only up to step t. We observe two interesting properties of the advantage function from this theorem. (1) *Disentanglement*: If (s, a) is reachable within t, then $\hat{A}^*(s, a) = A^\pi(s, a)$ remains invariant when we apply the theorem to any $\hat{t} > t$, suggesting that additional $\hat{A}_{t'}$'s do not interfere with existing ones. (2) *Additivity*: One intuitive way to understand the term $\sum_{t'} \gamma^{t'} \hat{A}_{t'}$ is to view it as the total effect from the sequence of actions, which is simply the sum of individual effects $\hat{A}_{t'}$. A similar property was also pointed out by Kakade and Langford [2002], where they showed how the consecutive sum of the advantage function is related to the expected return.

This theorem also suggests a natural way to estimate the advantage function directly from sampled trajectories $\tau_1, \tau_2, \ldots, \tau_N$ by minimizing the following constrained loss function

$$L(\theta) = \frac{1}{N} \sum_{\tau=\tau_1,\ldots,\tau_N} \left(\sum_{t=0}^{\infty} \gamma^t \left(r_t - \hat{A}_\theta \left(s_t, a_t \right) \right) \right)^2 \quad \text{s.t.} \sum_a \pi(a|s) \hat{A}_\theta(s, a) = 0, \tag{9}$$

In practice, the constraint can be forced upon a general function approximator f_θ, such as a neural network, by choosing $\hat{A}_\theta(s, a) = f_\theta(s, a) - \sum_{a \in \mathcal{A}} \pi(a|s) f_\theta(s, a)$. In our method, neither V nor Q is required a priori. Instead, the advantage function is directly modeled and estimated from data, and hence we name our method **Direct Advantage Estimation** (DAE). Unlike learning V or Q where one aims to find a mapping between states (state-action pairs) and its expected return, DAE tries to regress the *sum* of $A(s, a)$ along the trajectories towards the associated returns. This can be interpreted as learning to *distribute* the return rather than to *predict* it.

One limitation of this method, similar to the MC methods, is that we have to sample trajectories until they terminate. To overcome this, we show that the value function can be seamlessly integrated into our method, which enables us to perform updates similarly to the n-step bootstrapping method.

Theorem 2. Following the notations in Theorem 1, let $V_{\text{target}}(s)$ be the bootstrapping target and $\hat{V}(s)$ be the value function to be learned, and

$$L(\hat{A}, \hat{V}) = \mathbb{E} \left[\left(\sum_{t'=0}^{t-1} \gamma^{t'} (r_{t'} - \hat{A}_{t'}) + \gamma^t V_{\text{target}}(s_t) - \hat{V}(s_0) \right)^2 \right]. \tag{10}$$

If $\hat{V}^* = \arg\min_{\hat{V}} L(\hat{A}, \hat{V})$ and $p(s_0{=}s) > 0$, then for any $\hat{A} \in F_\pi$, we have

$$\hat{V}^*(s) = \mathbb{E} \left[\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} + \gamma^t V_{\text{target}}(s_t) \middle| s_0{=}s \right]. \tag{11}$$

Additionally, let $w_t(s) = \gamma^{2t} p(s_t{=}s)$ and $W_t(s) = \sum_{t'=0}^{t} w_{t'}(s)$. If $\hat{A}^* = \arg\min_{\hat{A} \in F_\pi} L(\hat{A}, \hat{V})$ and (s, a) is reachable within $t-1$, then for any $\hat{V} : \mathcal{S} \to \mathbb{R}$, we have

$$\begin{aligned} \hat{A}^*(s, a) = \sum_{t'=0}^{t-1} \frac{w_{t'}(s)}{W_{t-1}(s)} \Bigg(& \mathbb{E} \left[r_{t'} + \cdots + \gamma^{t-t'-1} r_{t-1} + \gamma^{t-t'} V_{\text{target}}(s_t) \middle| s_{t'}{=}s, a_{t'}{=}a \right] \\ & - \mathbb{E} \left[r_{t'} + \cdots + \gamma^{t-t'-1} r_{t-1} + \gamma^{t-t'} V_{\text{target}}(s_t) \middle| s_{t'}{=}s \right] \Bigg) \end{aligned} \tag{12}$$

Note that in Equation 12, the minimizer $\hat{A}^*$ depends only on the bootstrapping target V_{target} but not the estimated value function $\hat{V}$, and vice versa. One might notice that Equation 11 is actually the multi-step update for the value function. This suggests that we can update $\hat{V}$ iteratively by letting $V_{\text{target}} = \hat{V}_{k-1}$ and $\hat{V}_k(s) = \hat{V}^*(s) = \mathbb{E} \left[\sum_{t'=0}^{t-1} \gamma^{t'} r_{t'} + \gamma^t \hat{V}_{k-1}(s_t) \middle| s_0{=}s \right]$, which replicates multi-step TD learning. Similarly, Equation 12 can be understood as a multi-step estimation for the advantage function with bootstrapping.

In practice, we can sample n-step trajectories and use Equation 10 as the loss function to estimate both the advantage function and the value function simultaneously. However, naively applying the theorem this way can cause slow learning because the value function is learned only for the first state in each

trajectory and distant state-action pairs are heavily discounted. To mitigate this, we further utilize each trajectory by treating each sub-trajectory as a trajectory sampled from the same policy. For example, let $(s_0, a_0, \ldots, s_t)$ be a trajectory. Then we treat $(s_i, a_i, \ldots, s_t)$ for all $i \in \{0, \ldots, t-1\}$ also as a trajectory sampled from the same policy. This can be seen as repeated applications of Theorem 2, which leads us to the loss function

$$L_A(\theta, \phi) = \mathbb{E}\left[\sum_{t'=0}^{t-1}\left(\sum_{k=t'}^{t-1}\gamma^{k-t'}\left(r_k - \hat{A}_\theta(s_k, a_k)\right) + \gamma^{t-t'}V_{\text{target}}(s_t) - \hat{V}_\phi(s_{t'})\right)^2\right]. \quad (13)$$

In practice we may use a single approximator with parameters θ to model both $\hat{A}$ and $\hat{V}$ simultaneously as done by Wang et al. [2016b], which reduces the loss function to $L_A(\theta)$.

5 Experiments

We evaluate our method empirically by comparing the performance of policy optimization between DAE and GAE on three different sets of discrete control tasks. (1) A synthetic environment based on Figure 1 where the true advantage function and the expected return are known, (2) the MinAtar [Young and Tian, 2019] suite, a set of environments inspired by Atari games with similar dynamics but simpler observation space, and (3) the Arcade Learning Environment (ALE) [Bellemare et al., 2013], a set of tasks with diverse mechanisms and high-dimensional observation spaces.

5.1 Synthetic environment

We consider a finite variant of the environment shown in Figure 1 with state space $\mathcal{S} = \{s_1, ..., s_{128}\}$, and train agents using a simple actor-critic algorithm where in each iteration we (1) sample trajectories, (2) estimate the advantages of the sampled state-action pairs, and (3) perform a single policy gradient step weighted by the estimated advantages. For both methods, we use the same network architectures and hyperparameters to train the agents, see Appendix C for more details. We assess the quality of the advantage estimation through mean squared error (MSE) between the estimated and the true advantage function (Equation 5) on the sampled state-action pairs in each iteration. Additionally, we evaluate the learned policy by computing the true (undiscounted) expected return by $\mathbb{E}\left[\sum_i r_i\right] = \sum_i \pi(\mathtt{u}|s_i)$. Note that the optimal policy in this case is $\pi(\mathtt{u}|\cdot) = 1$ with expected return $\sum_i \pi(\mathtt{u}|s_i) = 128$.

The results in Figure 4 demonstrate that DAE can approximate the true advantage function accurately, while GAE struggles. The inability of GAE to accurately estimate the advantage function is caused by the variance of the n-step returns used in the estimates, which slowly decreases as the policy converges to the deterministic greedy policy. For policy optimization, we see that GAE performs better in the early phase of training, but is later surpassed by DAE. This is likely because estimates from GAE include unbiased estimates of n-step returns that are useful to policy optimization, while DAE relies entirely on the approximated function which is heavily biased at the beginning. As DAE becomes more accurate later in training, we see substantial gains in performance, which demonstrates a drawback of using high variance estimates from GAE.

5.2 MinAtar & Arcade Learning Environment (ALE)

We now test DAE's performance in the more challenging domains of the MinAtar and the ALE environments using PPO as the base algorithm. We use the PPO implementation and the tuned hyperparameters from Raffin et al. [2021], Raffin [2020]. A practical implementation of PPO with DAE can be found in Appendix B. For DAE, we tune two of the hyperparameters, namely the scaling coefficient of the value function loss and the number of epochs in each PPO iteration using the MinAtar environments, which are then fixed for the ALE experiments. Note that DAE has one less hyperparameter (λ) to tune because it does not rely on TD(λ). Additionally, we increase the number of parallel actors for both methods as it substantially speeds up PPO training in terms of wall-clock time and has become the predominant way to train PPO agents in complex environments [Rudin et al., 2022, Freeman et al., 2021]. We also test how network capacity impacts the performance of both methods by increasing the width of each layer in the baseline network (denoted "Wide"). For ALE, we further test a deep residual network architecture (denoted "Deep") based on Espeholt et al. [2018]. See Appendix C for a more detailed description of the network architectures and hyperparameters.

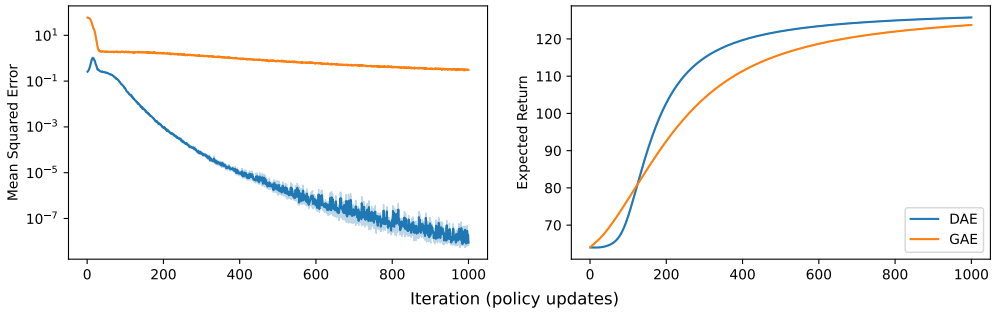

Figure 4: Results for the synthetic environment. Lines and shadings represent the mean and one standard error over 100 random seeds. Left: Mean squared error (in log scale) between the true and estimated advantage function on the state-action pairs sampled in each iteration. Right: The (undiscounted) expected return of the policy.

Each agent is trained independently for 10 million and 40 million in-game frames for the MinAtar and the ALE environments, respectively. We compare both methods on two metrics, (1) *Overall*: average undiscounted score of all training episodes, and (2) *Last*: average undiscounted score of the last 100 training episodes, as in Schulman et al. [2017]. The metrics reflect how fast an agent learns and the final performance of the learned policy, respectively.

From Table 1, we observe that, given the same network architecture, DAE outperforms GAE in all the MinAtar environments and most of the ALE environments on both metrics. In Figure 5, we present the GAE-normalized learning curves for each domain and architecture, where we also see a similar trend that DAE is much more sample efficient while achieving better final performance.[1] Results for individual environments can be found in Appendix E. Finally, we found that increasing the network capacity consistently improves DAE's performance while having less impact on GAE. This is likely due to the way the advantage function is directly represented in DAE using the network, such that network capacities should have a larger impact on the performance. Interestingly, we see significant improvements in the MinAtar experiments by switching to the Wide network for DAE but not GAE. Note that this gap cannot be explained by the increased representation power in the policy network alone, or we should also see significant improvements with GAE. Therefore, it suggests that the improvements are likely originated from the ability to more accurately approximate the advantage function. In the ALE experiments, we also observe such jumps in performance for some environments (see Appendix C), but for most environments, both DAE and GAE improved significantly. This can be attributed to the increased capacities in the policy networks, which benefit both methods.

6 Related Work

The advantage function was first introduced by Baird [1993] to tackle problems with fine-grained time steps, where Q-Learning could be slow. In policy optimization methods, the advantage function has also been widely used as an alternative to Q-functions to lower the variance of policy updates [Greensmith et al., 2004, Kakade and Langford, 2002, Sutton et al., 2000]. More recently, Wang et al. [2016b] developed a neural network architecture which can jointly estimate the value function and the advantage function to achieve better performance with DQN [Mnih et al., 2015]. Our work complements theirs by justifying the use of policy centered functions in estimating the advantage function, and showing how it can be combined with multi-step learning. Schulman et al. [2015b] proposed GAE, a method which combines TD(λ) and TD error to estimate the advantage function, and demonstrated strong empirical results in policy optimization.

The link between causality and credit assignment in RL has also been explored in Mesnard et al. [2020], where the authors used the idea of counterfactuals to construct a future-conditional value function to reduce the variance of policy gradient updates. Connections between causal effects and the

[1] Hard exploration environments such as Montezuma's Revenge are excluded in the normalization because the GAE baseline does not improve consistently.

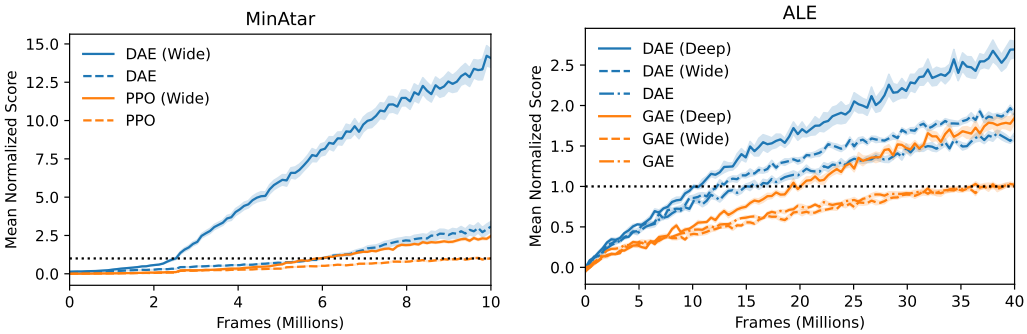

Figure 5: Mean GAE-normalized score for the MinAtar (left) and the ALE (right) experiments. The lines represent the average over all runs (50 and 10 random seeds for the MinAtar and the ALE, respectively) with each run corresponding to the average over all environments with one random seed, and the shadings correspond to one standard error of the mean. The scores are normalized based on GAE's (baseline network architecture) average score in the first 1% frames and the last 1% frames.

Table 1: Number of environments won by each method. The metrics were averaged across all runs before comparison. Results are counted as "Similar" if their 1 standard error interval overlaps. See Appendix E for detailed results.

Model Size	Method	Metric			
		Overall		Last	
		MinAtar	Atari	MinAtar	Atari
Baseline	GAE	0	10	0	11
	DAE	5	32	5	30
	Similar	0	7	0	8
Wide	GAE	0	8	0	9
	DAE	5	35	5	34
	Similar	0	6	0	6
Deep	GAE	-	6	-	8
	DAE	-	35	-	32
	Similar	-	8	-	9

advantage function were also studied by Corcoll and Vicente [2020], where they model the dynamics of environments in a hierarchical manner with a differently defined advantage function.

7 Conclusions and future directions

In this work, we tackled the problem of credit assignment in the RL setting. Inspired by previous studies in causality, we found that the advantage function can be interpreted as the causal effect of an action on the expected return. We then proposed DAE, a novel method that can directly model the advantage function and estimate it from data. Our experiments in three different domains demonstrated that DAE can converge to the true advantage function, improve the performance of policy optimization compared to GAE in most environments while also enjoying better scalability with network capacity.

Here we note some possible extensions to DAE. (1) Due to the difficulty of enforcing the normalizing constraint (Def. 1) in continuous domains, we currently restrict ourselves to discrete action space domains. One potential solution to this problem was outlined in Wang et al. [2016a] where they used a sampling method to approximate the normalization step for continuous action spaces. (2) In this work we assumed the data is on-policy, an interesting extension would be to use importance sampling techniques such as Retrace [Munos et al., 2016] or V-Trace [Espeholt et al., 2018] to extend DAE to off-policy settings.

References

L. C. Baird. Advantage updating. Technical report, WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.

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, jun 2013.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

O. Corcoll and R. Vicente. Disentangling causal effects for hierarchical reinforcement learning. *arXiv preprint arXiv:2010.01351*, 2020.

S. De and S. Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33:19964–19975, 2020.

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.

C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

J. Y. Halpern and C. Hitchcock. Graded causation and defaults. *The British Journal for the Philosophy of Science*, 66(2):413–457, 2015.

P. W. Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81 (396):945–960, 1986.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

T. Mesnard, T. Weber, F. Viola, S. Thakoor, A. Saade, A. Harutyunyan, W. Dabney, T. Stepleton, N. Heess, A. Guez, et al. Counterfactual credit assignment in model-free reinforcement learning. *arXiv preprint arXiv:2011.09464*, 2020.

M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

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.

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.

R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 1054–1062, 2016.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

J. Pearl. *Causality*. Cambridge University Press, 2009.

A. Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

D. B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of educational Psychology*, 66(5):688, 1974.

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

B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

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, 2015a.

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

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

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

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016a.

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, 2016b.

C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, 1989.

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

K. Young and T. Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 4 for the theory and Section 5 for the empirical results.
   (b) Did you describe the limitations of your work? [Yes] We address the limitations in Section 7
   (c) Did you discuss any potential negative societal impacts of your work? [No] Reinforcement learning is a powerful framework for task automation which may lead to malicious uses. However, our work focuses on a theoretical aspect of the framework, and hence we expect minimal societal impact.
   (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] See Section 4
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A
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] See Section 5 and Appendix C
   (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 C
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? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (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 Proofs

Throughout this section, we use $p(s_t{=}s, a_t{=}a)$ to denote the probability of the state-action pair at time step t being equal to (s, a), and the probability of a trajectory by $p(\tau) = p(s_0, a_0, s_1, a_1, ...)$.

A.1 Proof of Proposition 1

Proof. By definition

$$Q^\pi(s_t, a_t) = \mathbb{E}\left[\sum_{k\geq t}\gamma^{k-t} r_k \middle| s_t, a_t\right] \tag{14}$$

$$= \mathbb{E}\left[\sum_{k=t}^{t'-1}\gamma^{k-t} r_k \middle| s_t, a_t\right] + \gamma^{t'-t}\,\mathbb{E}\left[\sum_{k\geq t'}\gamma^{k-t'} r_k \middle| s_t, a_t\right] \tag{15}$$

$$= \mathbb{E}\left[\sum_{k=t}^{t'-1}\gamma^{k-t} r_k \middle| s_t, a_t\right] + \gamma^{t'-t}\,\mathbb{E}\left[V(s_{t'})|s_t, a_t\right] \tag{16}$$

Similarly, we have

$$V^\pi(s_t) = \mathbb{E}\left[\sum_{k=t}^{t'-1}\gamma^{k-t} r_k \middle| s_t\right] + \gamma^{t'-t}\,\mathbb{E}\left[V(s_{t'})|s_t\right] \tag{17}$$

If $\mathbb{E}\left[V(s_{t'})|s_t, a_t\right] = \mathbb{E}\left[V(s_{t'})|s_t\right]$, then

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{18}$$

$$= \mathbb{E}\left[\sum_{k=t}^{t'-1}\gamma^{k-t} r_k \middle| s_t, a_t\right] - \mathbb{E}\left[\sum_{k=t}^{t'-1}\gamma^{k-t} r_k \middle| s_t\right] \tag{19}$$

□

A.2 Proof of Theorem 1

Proof. For simplicity, we denote $\hat{A}$ by f. Notice that the condition $f \in F_\pi$ is equivalent to the equality constraint

$$\sum_{a\in\mathcal{A}} \pi(a|s) f(s, a) = 0 \quad \forall s \in \mathcal{S}. \tag{20}$$

Using the method of Lagrange multipliers, we have the Lagrangian

$$\mathcal{L} = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(G(\tau) - \sum_{t'=0}^{t}\gamma^{t'} f_{t'}\right)^2\right] + \sum_{s\in\mathcal{S}}\lambda_s \sum_{a\in\mathcal{A}}\pi(a|s) f(s, a) \tag{21}$$

with the shorthand $f_{t'} = f(s_{t'}, a_{t'})$. If we differentiate it with respect to $f(s', a')$, then

$$\frac{\partial\mathcal{L}}{\partial f(s', a')} = -2\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(G(\tau) - \sum_{t'=0}^{t}\gamma^{t'} f_{t'}\right)\left(\sum_{k=0}^{t}\gamma^k \frac{\partial f_k}{\partial f(s', a')}\right)\right] + \lambda_{s'}\pi(a'|s') \tag{22}$$

$$= -2\sum_{k=0}^{t}\gamma^k \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(G(\tau) - \sum_{t'=0}^{t}\gamma^{t'} f_{t'}\right)\frac{\partial f_k}{\partial f(s', a')}\right] + \lambda_{s'}\pi(a'|s') \tag{23}$$

Note that

$$\frac{\partial f_k}{\partial f(s', a')} = \mathbb{I}(s_k = s, a_k = a) = \begin{cases} 1, & \text{if } (s_k, a_k) = (s', a') \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

Hence,

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\right)\frac{\partial f_k}{\partial f(s',a')}\right] \tag{25}$$

$$=\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\right)\mathbb{I}(s_k=s,a_k=a)\right] \tag{26}$$

$$=p(s_k=s',a_k=a')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k=s',a_k=a'\right] \tag{27}$$

If we sum over all $a'\in\mathcal{A}$, since $p(s_k{=}s',a_k{=}s')=p(s_k{=}s')\pi(a'|s')$, then

$$\sum_{a'\in\mathcal{A}}\frac{\partial\mathcal{L}}{\partial f(s',a')}=-2\sum_{k=0}^{t}\gamma^k p(s_k=s')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k=s'\right]+\lambda_{s'}=0 \tag{28}$$

which implies

$$\lambda_{s'}=2\sum_{k=0}^{t}\gamma^k p(s_k=s')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k=s'\right]. \tag{29}$$

Substituting this back in Equation 22, we get, assuming $\pi(a'|s')>0$,

$$\sum_{k=0}^{t}\gamma^k p(s_k{=}s')\left(\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k{=}s',a_k{=}a'\right]-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k{=}s'\right]\right)=0 \tag{30}$$

From the Markov property, we have

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k=s',a_k=a'\right]-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t}\gamma^{t'}f_{t'}\middle|s_k=s'\right] \tag{31}$$

$$=\gamma^k\left(\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t-k}\gamma^{t'}f_{t'}\middle|s_0=s',a_0=a'\right]-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[G(\tau)-\sum_{t'=0}^{t-k}\gamma^{t'}f_{t'}\middle|s_0=s'\right]\right) \tag{32}$$

$$=\gamma^k\left(Q(s',a')-f(s',a')-V(s')\right) \tag{33}$$

If s' is reachable within t (i.e., $p(s_k=s')>0$ for some k), then

$$f(s',a')=Q(s',a')-V(s') \tag{34}$$

□

A.3 Proof of Theorem 2

Proof. We denote $\hat{A}$ by f and V_{target} by U, and use the method of Lagrange multipliers. Consider the Lagrangian

$$\mathcal{L}=\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\right)^2\right]+\sum_{s\in\mathcal{S}}\lambda_s\sum_{a\in\mathcal{A}}\pi(a|s)f(s,a) \tag{35}$$

Let's first consider the minimum for $\hat{V}$,

$$\frac{\partial\mathcal{L}}{\partial\hat{V}(s')}=-2p(s_0=s')\sum_{\tau}p(\tau|s_0=s')\left(\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s')\right) \tag{36}$$

$$=-2p(s_0=s')\left(\mathbb{E}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle|s_0=s'\right]\right)=0 \tag{37}$$

since $\mathbb{E}\left[\sum_{t'=0}^{t-1}\gamma^{t'}f_{t'}\middle| s_0=s'\right]=0$. If $p(s_0=s')>0$, then

$$\hat{V}(s')=\mathbb{E}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r_{t'}+\gamma^t U(s_t)\middle| s_0=s'\right] \tag{38}$$

which completes the first part of the proof. Next, we prove the second part of the theorem regarding f. Similar to the proof of Theorem 1, we consider

$$\frac{\partial\mathcal{L}}{\partial f(s',a')}=-2\sum_{k=0}^{t-1}\gamma^k\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\right)\left(\frac{\partial f_k}{\partial f(s',a')}\right)\right]+\lambda_{s'}\pi(a'|s')=0 \tag{39}$$

.

Following the proof of Theorem 1, we know that

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\left(\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\right)\left(\frac{\partial f_k}{\partial f(s',a')}\right)\right] \tag{40}$$

$$=p(s_k{=}s',a_k{=}a')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s',a_k{=}a'\right] \tag{41}$$

We substitute this back into Equation 39 and sum over the action space, which results in

$$\sum_{a'\in\mathcal{A}}\frac{\partial\mathcal{L}}{\partial f(s',a')}=-2\sum_{k=0}^{t-1}\gamma^k p(s_k{=}s')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s'\right]+\lambda_{s'}=0 \tag{42}$$

In other words,

$$\lambda_{s'}=2\sum_{k=0}^{t-1}\gamma^k p(s_k{=}s')\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s'\right] \tag{43}$$

Substituting this back again into Equation 39, we have

$$\begin{aligned}\sum_{k=0}^{t-1}\gamma^k p(s_k{=}s',a_k{=}a')\Bigg(&\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s',a_k{=}a'\right]\\&-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s'\right]\Bigg)=0.\end{aligned} \tag{44}$$

Using the Markov property, the term in the parentheses can be simplified to

$$\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s',a_k{=}a'\right] \tag{45}$$

$$\begin{aligned}&-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r'_{t'}+\gamma^t U(s_t)-\hat{V}(s_0)\middle| s_k{=}s'\right]\\&=\gamma^k\Bigg(\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-k-1}\gamma^{t'}r_{t'}+\gamma^{t-k}U(s_{t-k})\middle| s_0{=}s',a_0{=}a'\right]\\&\quad-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-k-1}\gamma^{t'}r_{t'}+\gamma^{t-k}U(s_{t-k})\middle| s_0{=}s'\right]-f(s',a')\Bigg)\end{aligned} \tag{46}$$

Finally, if (s',a') is reachable within $t-1$, then

$$f(s',a')=\frac{1}{\sum_{k=0}^{t-1}w_k(s')}\sum_{k=0}^{t-1}w_k(s')\Bigg(\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-k-1}\gamma^{t'}r_{t'}+\gamma^{t-k}U(s_{t-k})\middle| s_0{=}s',a_0{=}a'\right] \tag{47}$$

$$-\mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t'=0}^{t-k-1}\gamma^{t'}r_{t'}+\gamma^{t-k}U(s_{t-k})\middle| s_0{=}s'\right]\Bigg), \tag{48}$$

with the shorthand $w_k(s')=\gamma^{2k}p(s_k{=}s')$. □

B Integrating PPO with DAE

Algorithm 1 shows a practical implementation of PPO with DAE. The PPO clipping loss is

$$L_\pi = \mathbb{E}\left[\min\left(\frac{\pi_\theta(a|s)}{\mu(a|s)}\hat{A}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\mu(a|s)}, 1-\epsilon, 1+\epsilon\right)\hat{A}(s,a)\right)\right], \tag{49}$$

where π_θ is the policy that is being optimized, μ is the sampling policy and $\hat{A}(s,a)$ is the estimated advantage function. In practice, an entropy loss $H_\pi = \sum_{a\in\mathcal{A}} \pi(a|s)\log\pi(a|s)$ is also added to L_π to encourage exploration. Note that, unlike the original PPO which samples mini-batches of frames, we sample on a trajectory-by-trajectory basis. For example, assume the batch size is 256 and $n = 128$ for the backup horizon, then each batch would contain 2 128-step trajectories.

Algorithm 1 PPO with DAE (shared network)

Initialize the $\hat{A}$, $\hat{V}$ and the policy π with θ
while not converged **do**
  $D = \{\}$
  for $i = 1, \ldots, N_{\text{actors}}$ **do**
    Sample n-step trajectory $\tau_i = (s_0, a_0, r_0, \ldots, s_n)$ with policy π
    $D \leftarrow D \cup \{\tau_i\}$
  end for
  $U \leftarrow \texttt{stop_gradient}(\hat{V})$
  for $i = 1, \ldots, N_{\text{updates}}$ **do**
    Sample a mini-batch of trajectories B from D
    Compute $\hat{A}(s,a)$ for all $(s,a) \in B$
    Compute PPO clipping loss L_π with $\texttt{stop_gradient}(\hat{A}(s,a))$
    Compute L_A with $\hat{A}$, $\hat{V}$ and U by Equation 13
    $\theta \leftarrow \theta - \alpha\nabla(L_\pi + \beta_V L_A + \beta_{\text{ent}} H_\pi)$
  end for
end while

C Experiment details

C.1 Computational resources

All the experiments were performed on an internal cluster of NVIDIA A100 GPUs. Training a MinAtar agent in a single environment takes less than 30 minutes (wall-clock time). Training an ALE agent on a single environment may take up to 9 hours (wall-clock time) depending on the network capacity.

C.2 Variation Study

We train a DAE baseline PPO agent with the hyperparameters described below and save the states encountered by each actor along with a checkpoint of the policy at the start of each PPO iteration for 50 iterations. Afterwards, we sample uniformly 1024 states from the set of collected states. For these states, we perform 128 MC rollouts (up to 512 steps) for every action (the MinAtar Breakout environment has 3 actions) to estimate the true $Q(s, a)$, $V(s)$ and $A(s, a)$ for every policy.

C.3 Synthetic environment

To avoid giving DAE an unfair advantage because it could learn a function that ignores the input (since the advantage function is the same for each state), we randomly swap the rewards of `u` and `d` at each state at the beginning of the experiment.

We summarize the hyperparameters in Table 2. The policy and the value/advantage network are modeled separately. We parameterize the policy using $\theta_{ij} \in R^{128\times 2}$, representing the logit of each action. θ_{ij} are initialized to 0 at the beginning of each experiment. The value/advantage network consists of MLPs of 2 hidden layers of size 256, and the input states are represented using one-hot encoding.

Table 2: Hyperparameters for the synthetic experiment.

Parameter	Value	
	GAE	DAE
Discount γ	0.99	
$N_{\text{iterations}}$	1000	
Optimizer	Adam	
Learning rate (value/policy)	0.001/0.01	
Adam β	(0.9, 0.999)	
Adam ϵ	10^{-3}	
Sample trajectories per iteration	4	
Value gradients per iteration	4	
Policy gradients per iteration	1	
Batch Size	Whole dataset	
GAE λ	0.95	—

C.4 MinAtar & ALE

C.4.1 Preprocessing

MinAtar. We turn off sticky action and difficulty ramping to further simplify the environments.

ALE. We follow the standard preprocessing procedures from Mnih et al. [2015], except for the max frames per episode where we used the default value from the gym implementation [Brockman et al., 2016]. See Table 3 for a summary of the parameters.

C.4.2 Network architecture

For the MinAtar and the ALE experiments, we summarize the baseline network architectures in Figure 6. For the wide network experiments, we simply multiply the numbers of channels and widths

Table 3: ALE preprocessing parameters.

Parameter	Value
Grey-scaling	True
Observation down-sampling	84×84
Frame stack	4
Frame skip	4
Reward clipping	$[-1, 1]$
Terminal on loss of life	True
Max frames per episode	400K

of each hidden layer by 4 (MinAtar) or 2 (ALE). We follow Espeholt et al. [2018] for the Deep network architecture used in the ALE experiments, except we multiply the number of channels of each layer by 4 and increase the width of the last fully connected layer to 512. Additionally, we use SkipInit De and Smith [2020] to initialize the residual connections, which we found to stabilize learning.

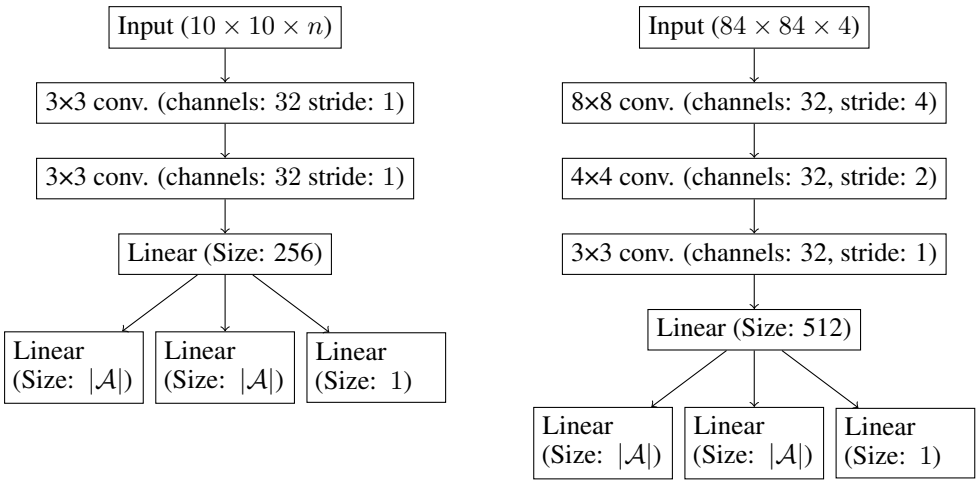

Figure 6: The baseline network architectures for the MinAtar experiments (left) and the ALE experiments (right). Each hidden layer is followed by a ReLU activation. The three output streams correspond to $A(s, a)$, $\pi(a|s)$, and $V(s)$, respectively.

C.4.3 PPO hyperparameters

We use the tuned PPO hyperparameters from Raffin [2020] for GAE. As for DAE, we additionally tune the number of epochs per iteration {4, 6, 8} and the β_V coefficient {0.5, 1, 1.5} using the MinAtar environments. We also increase the number of parallel actors from 8 to 1024 to speed up training, the number 1024 was chosen to maximize GPU memory usage in the ALE experiments. Among the tuned hyperparameters, we found that having a large number of parallel actors is the most important one. This is likely because DAE relies entirely on the network to approximate the advantage function, so having a huge batch of data at each PPO iteration is critical to having reliable estimates. Aside from the above-mentioned hyperparameters, we have also tried using separate networks for the policy and the advantage function, or replacing PPO clipping with KL-divergence penalties, but found them less effective than the original PPO algorithm.

We summarize the final set of hyperparameters in Table 4. The learning rate and the PPO clipping ϵ are linearly annealed towards 0 throughout training.

Table 4: Hyperparameters for the ALE experiments.

Parameter	Value	
	GAE	DAE
Discount γ	0.99	
N_{actor}	1024	
N_{steps}	128	
Optimizer	Adam	
Learning rate	0.00025	
Adam β	(0.9, 0.999)	
Adam ϵ	10^{-5}	
N_{Epochs}	4	6
Batch Size	256	
PPO Clipping ϵ	0.1	
β_V	0.5	1.5
β_{ent}	0.01	
GAE λ	0.95	—
Weight Initialization	orthogonal	

D Ablation Study

In addition to the GAE baseline, here we consider two additional baselines to demonstrate the effectiveness of DAE.

1. Indirect: We learn both Q and V separately by minimizing the n-step bootstrapping losses.

$$L_Q = \mathbb{E}\left[\left(\hat{Q}(s,a) - \left(r_0 + ... + \gamma^{n-1} r_{n-1} + \gamma^n V_{\text{target}}(s_n)\right)\right)^2 \middle| s_0{=}s, a_0 = a\right] \tag{50}$$

$$L_V = \mathbb{E}\left[\left(\hat{V}(s) - \left(r_0 + ... + \gamma^{n-1} r_{n-1} + \gamma^n V_{\text{target}}(s_n)\right)\right)^2 \middle| s_0{=}s\right] \tag{51}$$

   where $\hat{Q}$ and $\hat{V}$ are the learned Q-function and the value function. $\hat{Q}$ and $\hat{V}$ can then be used to estimate the advantage function via $\hat{A}(s,a) = \hat{Q}(s,a) - \hat{V}(s)$. This baseline demonstrates the effectiveness of learning the advantage function directly.

2. Duel: Based on Wang et al. [2016b], we slightly modify the original loss function to extend to the n-step setting. The new loss function is now

$$L = \mathbb{E}\left[\left(\hat{V}(s) + \hat{A}(s,a) - \left(r_0 + ... + \gamma^{n-1} r_{n-1} + \gamma^n V_{\text{target}}(s_n)\right)\right)^2 \middle| s_0{=}s, a_0 = a\right], \tag{52}$$

   where $\hat{V}$ and $\hat{A}$ are being learned. Essentially, this differs from the DAE loss in whether we sum over the advantage function over time steps. Unlike the original dueling architecture, we use the learned policy instead of the uniform policy to enforce the π-centered constraint. This baseline demonstrates the effectiveness of the DAE loss (sum over advantage functions).

We compare their performance in policy optimization using the MinAtar suite with the settings and DAE hyperparameters (baseline network) described in Appendix C. We show the learning curves for individual environments and the normalized curves in Figure 7 and Figure 8. Our results show that the Indirect method performs worse than both DAE and Duel, suggesting that explicitly modelling the advantage function can be beneficial. Furthermore, by comparing the results from DAE and Duel, we see substantial gains by switching to the DAE loss, even surpassing the performance of GAE. This suggests that jointly estimating the advantage function across time steps is crucial to the performance of DAE.

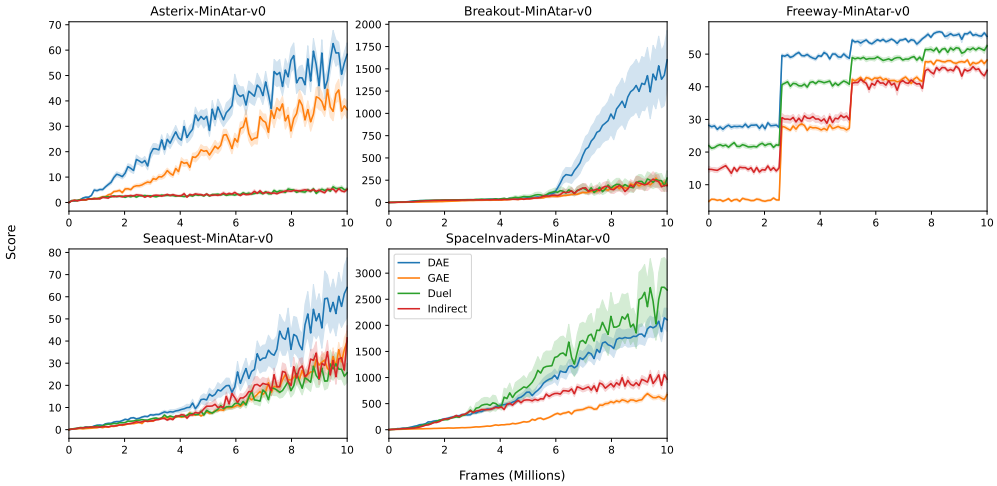

Figure 7: Learning curves for the raw scores in the MinAtar experiments. Lines and shadings represent the average and one standard error over 50 runs.

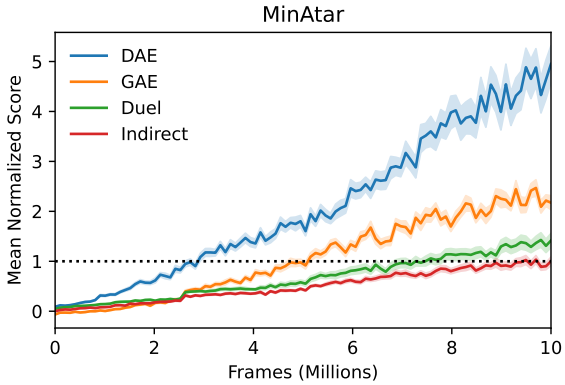

Figure 8: Normalized learning curves for the ablation study. Scores are normalized independently for each environment based on scores from the Indirect baseline before aggregated (50 seeds).

E Additional results

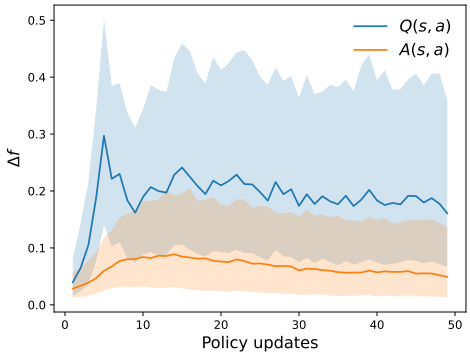

Figure 9: Variation of the advantage function and the Q-function from the MinAtar Asterix environment. See Appendix C for details.

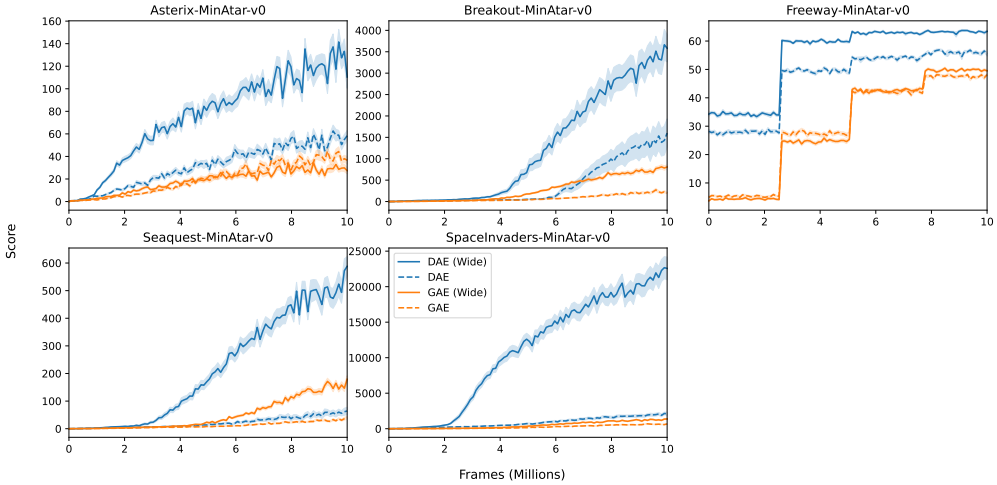

Figure 10: Learning curves for the raw scores in the MinAtar experiments. Lines and shadings represent the average and one standard error over 50 runs.

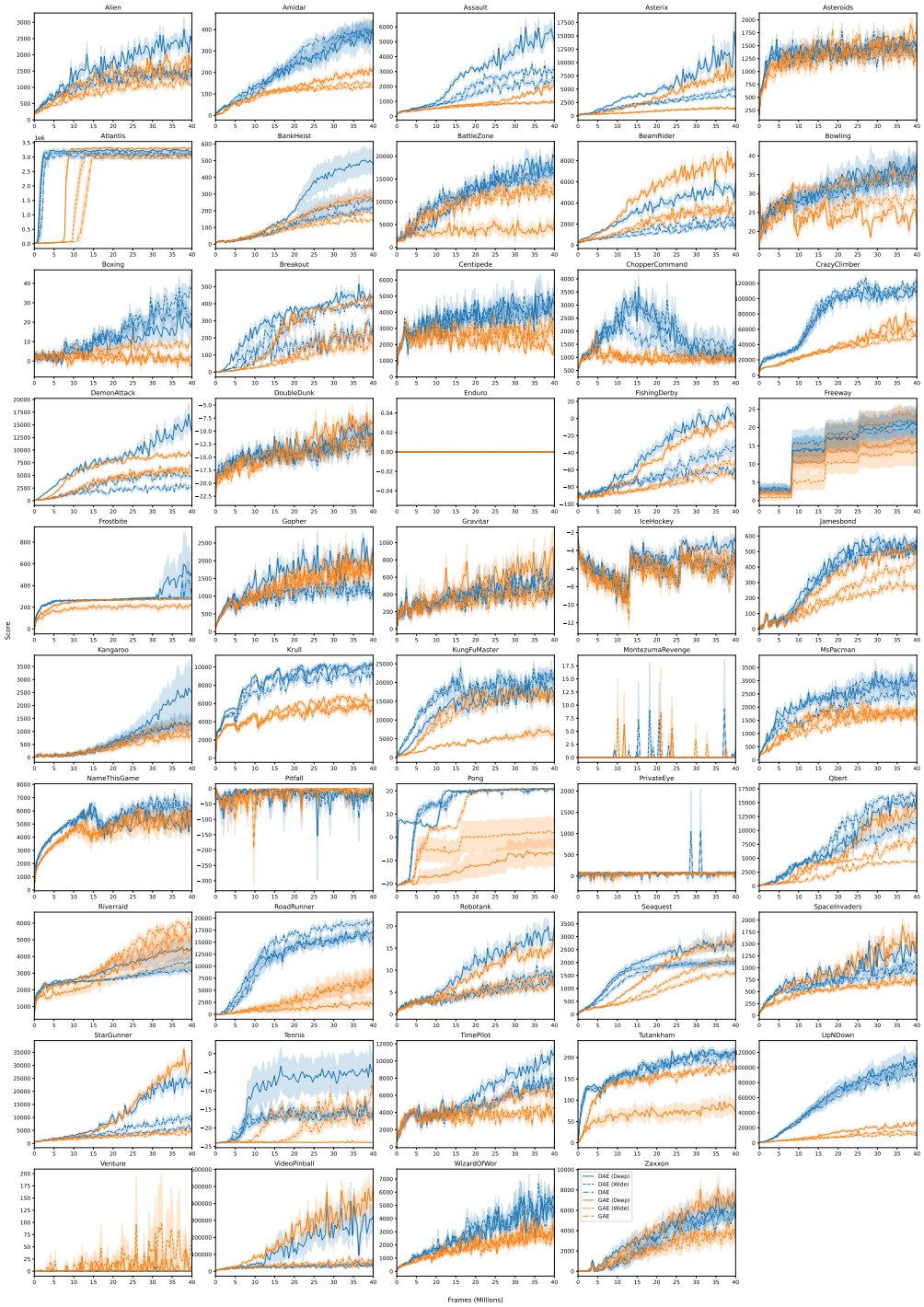

Figure 11: Learning curves for the raw scores in the ALE experiments. Lines and shadings represent the average and one standard error over 10 runs.

Table 5: Overall and last scores (defined in Section 5) on the MinAtar environments with baseline network. Numbers represent (mean)±(1 standard error of the mean).

Environment	Metric			
	Overall		Last	
	GAE	DAE	GAE	DAE
Asterix	6.6±0.1	10.5±0.1	27.4±0.6	40.7±1.0
Breakout	10.0±0.1	17.2±0.3	168.2±11.3	844.7±152.3
Freeway	30.2±0.3	46.5±0.2	47.5±0.2	55.9±0.2
Seaquest	4.4±0.1	6.3±0.3	26.4±1.9	45.9±7.2
SpaceInvaders	43.7±0.3	143.7±2.6	504.2±10.3	1675.3±140.1

Table 6: Overall and last scores (defined in Section 5) on the MinAtar environments with wide network. Numbers represent (mean)±(1 standard error of the mean).

Environment	Metric			
	Overall		Last	
	GAE	DAE	GAE	DAE
Asterix	7.3±0.1	18.0±0.1	22.1±0.4	89.3±1.9
Breakout	12.0±0.0	23.3±0.4	637.7±21.3	2409.5±190.5
Freeway	29.9±0.2	54.8±0.1	49.8±0.2	63.4±0.1
Seaquest	7.0±0.2	16.4±0.5	123.7±2.9	442.1±22.6
SpaceInvaders	62.6±0.4	276.0±2.0	1058.0±18.1	11743.3±315.9

Table 7: Overall and last scores (defined in Section 5) on the ALE environments with baseline network. Numbers represent (mean)±(1 standard error of the mean).

Environment	Metric			
	Overall		Last	
	GAE	DAE	GAE	DAE
Alien	809.9±9.5	1042.8±36.3	1165.7±24.3	1372.7±109.1
Amidar	90.5±0.9	201.8±13.4	129.4±3.6	394.6±38.6
Assault	653.1±4.0	1026.4±42.3	926.8±12.5	2205.1±115.4
Asterix	724.8±4.0	1696.8±51.7	1368.6±11.1	3750.1±168.4
Asteroids	1220.7±5.6	1235.1±11.6	1395.7±15.2	1392.3±19.4
Atlantis	161787.1±3594.5	886857.2±54327.4	2039346.4±30987.7	2888011.1±68326.9
BankHeist	82.1±2.0	111.1±15.3	144.2±4.6	257.8±61.0
BattleZone	9286.7±219.8	11440.0±358.6	12813.0±389.6	16302.0±628.5
BeamRider	1617.9±11.8	1027.0±33.1	2968.8±36.0	1729.9±56.2
Bowling	31.6±0.5	30.7±1.6	34.7±1.2	36.1±2.9
Boxing	5.7±0.2	13.8±1.8	8.5±0.3	25.9±3.1
Breakout	46.8±1.2	97.3±4.4	160.3±3.2	234.9±8.6
Centipede	2566.4±13.7	3238.3±76.3	2682.3±57.1	3915.8±222.1
ChopperCommand	1137.9±14.5	1772.5±108.7	1073.4±19.2	1587.3±137.8
CrazyClimber	33832.7±493.6	72051.1±2153.6	55877.1±1514.7	112319.5±1925.6
DemonAttack	2334.5±42.5	1373.5±46.2	5736.1±118.9	2477.2±108.6
DoubleDunk	-14.8±0.1	-15.5±0.6	-12.1±0.3	-12.5±1.0
Enduro	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
FishingDerby	-79.8±0.3	-72.6±1.3	-68.6±0.8	-60.4±2.3
Freeway	15.0±2.5	14.2±3.2	23.4±2.2	19.5±4.4
Frostbite	241.2±0.4	263.4±5.3	272.0±1.1	367.9±86.6
Gopher	1163.2±56.4	910.3±13.3	1666.0±90.5	1137.2±49.4
Gravitar	310.9±12.6	347.3±10.4	373.7±26.3	443.5±15.8
IceHockey	-5.9±0.1	-6.1±0.1	-4.5±0.1	-5.1±0.2
Jamesbond	138.1±6.2	237.4±6.2	277.1±13.2	507.8±6.9
Kangaroo	450.5±36.2	503.2±124.4	1022.0±97.2	1331.0±334.6
Krull	4573.8±35.9	7496.0±242.4	5642.2±72.8	9034.0±246.0
KungFuMaster	10614.3±124.6	14286.3±439.1	17389.1±250.8	20535.3±911.5
MontezumaRevenge	0.0±0.0	0.1±0.0	0.2±0.2	0.1±0.1
MsPacman	1148.4±18.4	1714.7±81.2	1674.8±20.2	2501.0±189.7
NameThisGame	4427.7±23.4	5013.9±64.4	5685.5±59.2	6016.3±91.8
Pitfall	-4.5±0.4	-7.5±0.6	-0.1±0.1	-12.0±6.7
Pong	8.9±0.2	12.5±0.5	20.9±0.0	20.7±0.1
PrivateEye	74.8±11.0	87.3±12.0	78.2±13.1	86.0±6.2
Qbert	1987.7±22.3	4622.6±250.9	4357.4±41.0	11119.6±1102.2
Riverraid	3347.2±39.2	2672.0±53.0	5133.2±93.0	3150.8±172.1
RoadRunner	2133.9±801.5	7445.8±572.7	6530.9±2194.0	16146.3±968.6
Robotank	3.9±0.1	4.1±0.3	6.0±0.2	6.9±0.9
Seaquest	669.0±26.9	1212.0±66.8	1580.4±68.8	2049.8±135.9
SpaceInvaders	474.1±1.5	614.6±29.4	665.4±8.4	914.9±68.6
StarGunner	2677.5±30.3	3127.2±103.1	4491.8±143.1	5849.2±232.6
Tennis	-23.1±0.1	-23.1±0.1	-16.6±0.2	-17.6±0.5
TimePilot	3417.1±23.9	4733.7±96.6	3758.4±75.5	7252.7±373.1
Tutankham	119.3±2.3	150.2±5.0	172.5±2.4	196.0±8.3
UpNDown	5714.1±89.4	19671.7±707.7	14288.2±516.8	85438.4±6152.3
Venture	0.1±0.1	0.0±0.0	0.0±0.0	0.0±0.0
VideoPinball	22791.1±208.0	20279.4±627.9	28256.7±1251.8	23958.6±1357.6
WizardOfWor	1605.4±33.7	2414.4±111.1	2489.5±110.4	4161.3±216.0
Zaxxon	1749.7±171.1	2520.5±339.9	3897.9±192.7	5612.2±540.0

Table 8: Overall and last scores (defined in Section 5) on the ALE environments with Wide network. Numbers represent (mean)±(1 standard error of the mean).

Environment	Metric			
	Overall		Last	
	GAE	DAE	GAE	DAE
Alien	957.5±14.4	1172.2±50.8	1452.3±39.5	1461.5±110.4
Amidar	99.1±1.3	191.3±18.0	150.2±5.8	391.8±46.3
Assault	648.3±6.0	1195.7±40.3	991.1±21.1	3002.4±78.2
Asterix	740.8±8.7	1960.4±77.1	1427.5±25.8	4782.3±516.5
Asteroids	1096.7±11.9	1333.9±14.2	1278.2±21.3	1549.0±18.7
Atlantis	187676.0±3832.8	1074670.4±53480.9	2261050.5±32146.1	3006634.5±63364.8
BankHeist	98.2±2.6	105.3±3.9	188.0±6.1	214.6±11.9
BattleZone	8959.6±211.3	10820.1±611.7	12155.0±285.5	15903.0±661.6
BeamRider	1839.0±16.4	1275.4±27.3	3540.0±23.2	2203.2±76.7
Bowling	26.3±0.5	29.9±1.9	28.5±0.7	34.4±2.6
Boxing	1.6±0.1	15.8±1.9	1.9±0.1	34.3±4.7
Breakout	62.4±2.3	155.3±2.1	217.4±8.4	383.4±4.1
Centipede	2426.7±24.3	3394.7±85.5	2482.7±45.4	4517.1±286.8
ChopperCommand	1094.5±40.5	2264.5±158.6	1024.7±33.5	2261.3±282.9
CrazyClimber	31296.7±476.3	77555.5±1712.7	49524.3±1617.6	118288.9±1709.6
DemonAttack	2452.2±33.6	2226.0±69.7	5902.1±78.8	4893.3±289.9
DoubleDunk	-15.4±0.1	-16.1±0.3	-12.8±0.3	-14.4±0.7
Enduro	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
FishingDerby	-75.3±1.0	-64.1±3.3	-52.9±2.1	-35.6±7.1
Freeway	8.2±2.3	15.6±2.9	13.4±3.7	21.8±3.8
Frostbite	246.1±0.8	262.5±2.4	277.1±2.4	284.6±6.3
Gopher	1159.6±54.1	980.6±31.5	1591.8±90.1	1225.9±102.2
Gravitar	327.9±18.6	378.5±15.3	450.6±35.5	524.8±30.9
IceHockey	-6.2±0.1	-5.8±0.1	-5.3±0.1	-4.5±0.1
Jamesbond	168.1±5.6	235.1±10.0	385.9±10.1	506.6±9.3
Kangaroo	533.7±32.7	474.2±85.9	1303.6±67.8	1346.8±248.9
Krull	4544.6±45.5	8340.4±58.4	5361.6±69.8	9953.9±81.8
KungFuMaster	9384.7±404.9	14780.2±424.2	15674.3±350.1	21483.3±658.4
MontezumaRevenge	0.1±0.0	0.1±0.0	0.3±0.2	0.0±0.0
MsPacman	1338.9±9.3	2166.2±138.8	1828.2±17.9	3065.3±278.9
NameThisGame	4297.1±22.8	5183.5±98.8	5303.2±97.2	6250.2±171.7
Pitfall	-4.7±0.4	-9.0±1.1	0.0±0.0	-5.7±2.2
Pong	-4.3±5.1	12.5±0.5	2.3±6.3	20.7±0.1
PrivateEye	44.5±13.2	48.2±11.4	61.6±11.7	69.6±8.8
Qbert	2945.0±37.0	6190.1±156.5	7748.5±182.2	16239.6±361.3
Riverraid	3623.1±65.2	2723.5±78.2	5830.5±116.7	3630.6±379.1
RoadRunner	2006.4±631.7	9600.6±406.5	6461.3±1517.3	18767.2±484.3
Robotank	4.7±0.1	4.8±0.3	7.5±0.2	9.1±0.9
Seaquest	793.3±21.5	1285.6±49.1	2100.6±79.6	1967.2±126.8
SpaceInvaders	485.1±1.3	658.5±28.0	699.9±11.7	1098.3±73.4
StarGunner	2319.1±32.2	4286.1±195.1	3752.2±121.5	9123.1±510.7
Tennis	-23.7±0.0	-23.2±0.1	-20.4±0.9	-17.6±0.8
TimePilot	3514.2±17.4	4813.9±67.7	3902.5±58.4	7649.6±288.4
Tutankham	123.7±1.5	159.8±2.7	180.5±2.2	209.0±4.3
UpNDown	4968.0±70.5	20478.4±881.1	11014.5±246.0	94550.9±6967.9
Venture	21.8±21.7	0.1±0.1	42.4±42.4	0.2±0.2
VideoPinball	30055.3±555.4	24036.4±704.1	46653.9±2099.4	32161.8±1625.1
WizardOfWor	1622.4±37.2	2665.1±77.0	2591.7±100.3	5057.0±244.2
Zaxxon	1661.7±425.7	2048.2±395.2	3496.6±720.5	5275.2±423.5

Table 9: Overall and last scores (defined in Section 5) on the ALE environments with Deep network. Numbers represent (mean)±(1 standard error of the mean).

Environment	Metric			
	Overall		Last	
	GAE	DAE	GAE	DAE
Alien	1151.3±9.4	1501.4±59.5	1735.9±42.2	2396.3±138.1
Amidar	115.4±3.8	192.7±22.8	205.8±15.5	362.0±55.2
Assault	788.5±8.7	1587.8±48.8	1862.4±81.2	4963.1±280.4
Asterix	2010.7±58.3	3403.3±80.7	7239.5±498.9	11861.7±1190.4
Asteroids	1309.5±8.3	1335.8±14.6	1581.1±27.7	1531.2±28.1
Atlantis	250696.5±3918.9	1220135.9±62706.7	2649450.3±24887.5	3082321.8±64162.3
BankHeist	137.3±14.4	180.4±11.8	279.2±29.0	494.1±74.9
BattleZone	3373.1±872.2	10584.5±775.1	4180.0±1298.8	18410.0±1261.4
BeamRider	3578.3±39.5	2801.9±66.4	7597.5±119.8	5179.8±182.5
Bowling	23.9±0.2	31.9±2.2	24.2±0.2	37.1±3.3
Boxing	1.5±0.0	10.0±3.5	1.7±0.1	19.3±6.6
Breakout	108.2±4.3	193.5±3.4	410.2±6.6	440.6±6.2
Centipede	2048.1±17.2	3407.9±89.5	1897.2±28.4	4505.4±168.7
ChopperCommand	1006.2±23.7	2177.2±113.3	973.0±28.0	2135.1±200.6
CrazyClimber	37238.2±1126.0	71207.8±3992.4	66005.2±3164.9	109013.9±4292.6
DemonAttack	4392.1±59.0	5906.1±198.6	9064.1±95.4	13841.7±922.5
DoubleDunk	-13.7±0.3	-15.0±0.8	-8.5±0.7	-12.7±1.4
Enduro	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
FishingDerby	-49.2±1.1	-36.1±3.9	-8.7±1.6	8.4±3.7
Freeway	11.2±2.2	14.5±2.8	16.0±2.5	20.9±3.7
Frostbite	184.8±18.7	283.9±22.5	207.9±23.0	494.1±200.5
Gopher	1222.7±73.2	1444.7±103.7	1721.5±137.2	2000.1±143.8
Gravitar	482.7±32.6	397.5±18.5	710.5±61.9	581.8±41.3
IceHockey	-6.5±0.0	-4.8±0.3	-6.0±0.1	-3.0±0.4
Jamesbond	230.5±12.9	267.5±9.3	489.4±16.3	530.4±12.2
Kangaroo	351.5±64.0	585.4±156.3	879.4±170.2	2453.9±899.2
Krull	5121.2±51.3	8464.9±48.6	6210.1±101.9	9847.3±95.1
KungFuMaster	3468.0±187.2	10871.2±999.5	6535.3±1089.9	16581.3±1497.8
MontezumaRevenge	0.0±0.0	0.1±0.0	0.0±0.0	0.4±0.4
MsPacman	1386.4±142.1	2150.3±72.3	1774.8±194.8	3057.2±219.6
NameThisGame	4101.7±458.2	4654.8±103.7	4757.8±577.4	5109.2±164.8
Pitfall	-10.3±1.2	-8.2±0.4	-6.3±1.3	-6.3±1.0
Pong	-14.3±3.5	17.3±0.1	-6.6±5.7	20.9±0.0
PrivateEye	53.2±0.8	53.7±5.1	54.0±3.2	55.0±6.3
Qbert	4165.5±414.2	5563.4±197.3	13009.7±1212.3	15177.2±399.3
Riverraid	2740.5±542.5	3206.6±212.4	4392.3±1105.7	4455.4±610.3
RoadRunner	867.4±448.3	8604.9±1041.4	2153.5±1231.5	16665.6±795.0
Robotank	7.4±0.1	8.7±0.7	14.0±0.3	17.0±1.2
Seaquest	1111.5±42.7	1553.3±64.7	2776.3±249.5	2723.7±287.0
SpaceInvaders	755.2±9.6	755.8±20.9	1421.9±58.5	1269.5±75.3
StarGunner	7281.1±222.2	7400.8±253.4	29961.4±1274.0	23087.6±1421.8
Tennis	-23.9±0.0	-16.3±3.4	-23.9±0.0	-6.6±5.4
TimePilot	4548.1±143.9	5942.2±156.1	6523.9±377.6	10510.2±313.1
Tutankham	57.6±12.9	160.8±3.4	85.9±19.1	212.5±6.5
UpNDown	7692.6±254.0	23790.1±2304.3	22454.7±1222.5	101203.3±12747.0
Venture	6.9±4.6	0.0±0.0	12.1±8.1	0.0±0.0
VideoPinball	133908.8±4973.8	66261.9±10063.3	348512.7±23846.3	234364.9±55812.2
WizardOfWor	1641.1±107.4	2537.6±68.1	2599.3±225.9	4537.2±178.4
Zaxxon	3101.4±383.3	3011.4±509.4	6982.3±695.0	6438.6±1042.5