Generalized Policy Updates for Policy Optimization

Saurabh Kumar[1*†], **Zafarali Ahmed**[2*‡], **Robert Dadashi**[1*],

Dale Schuurmans[1], **Marc G. Bellemare**[1],

[1]Google Brain [2]Mila, McGill University

szk@stanford.edu, {zaf,dadashi,bellemare,schuurmans}@google.com

Abstract

In reinforcement learning, policy gradient methods follow the gradients of well-defined objective functions. While the standard on-policy policy gradient maximizes the expected return objective, recent work has shown that it suffers from accumulation points. By contrast, other policy search methods have demonstrated better sample efficiency and stability. In this paper, we introduce a family of generalized policy updates that are not in general the gradients of an objective function. We establish theoretical requirements under which members of this family are guaranteed to converge to an optimal policy. We study two natural instances of this family, expected actor-critic (EAC) and argmax actor-critic (AAC), and relate them to existing successful methods. We apply both EAC and AAC in experiments on synthetic gridworld domains and Atari 2600 games to demonstrate that they are both sample efficient and stable, and outperform both on-policy policy gradient and a value-based baseline. Our results provide further evidence that following the gradient of the expected return is not necessarily the most efficient way to obtain an optimal policy.

1 Introduction

In reinforcement learning (RL), policy gradient methods optimize a differentiable policy, parametrized by $\theta \in \mathbb{R}^d$, that maps states to distributions over actions. The policy is optimized with respect to an objective function $J : \mathbb{R}^d \to \mathbb{R}$ by updating the policy parameters in the direction of the gradient $\nabla_\theta J(\theta)$. The canonical objective function $J(\theta)$ consists in finding a policy that maximizes the expected discounted sum of rewards obtained when following that policy. Methods such as REINFORCE [36] follow simple unbiased estimates of the policy gradient $\nabla_\theta J(\theta)$, while actor-critic schemes also estimate a value function and use this estimate to improve their policy [31; 20; 14]. Policy gradient methods are explicitly designed with the objective $J(\theta)$ in mind, and they have been successfully applied to a number of problems [32; 21; 19].

Informally, developments in policy gradient methods have been driven to achieve two desired properties:

1. Sample efficiency: we would like methods that require few samples to produce a near-optimal policy (related to the speed of convergence), and
2. Stability: we would like methods that are robust to different random seeds and gradient estimation error.

[*]Denotes equal contribution

[†]Work done as part of the Google AI Residency, now at Stanford University.

[‡]Work done while at Mila and Google Brain, now at DeepMind.

Optimization Foundations for Reinforcement Learning Workshop at NeurIPS 2019, Vancouver, Canada.

On-policy policy gradient methods collect a batch of experience from the current policy π_θ, estimate the policy gradient using this data, and apply the gradient estimate to the policy's parameters. This process is stable but not sample efficient, since it can only use data obtained from the current policy. The presence of accumulation points or plateaus also contributes to high sample complexity and slow convergence [17; 3; 13]. At plateaus, many samples are required to correctly estimate the gradient direction, making it difficult to find an optimal policy.

In the policy gradient literature, a standard approach to improve the sample efficiency of policy optimization is to construct surrogate objectives that approximate, but are distinct from, the canonical objective. Maximum entropy reinforcement learning augments the expected discounted return objective by adding the entropy of the policy [37]. One potential consequence of maximizing entropy is to prevent the policy from converging to locally optimal deterministic policies. Soft Actor-Critic [15] combines maximum entropy RL with off-policy learning to achieve both sample efficiency and stability. Alternatively, natural gradient methods [18; 25] and related algorithms such as TRPO and PPO [28; 30] constrain how much the policy changes with each update to attain stable and steady improvement of the policy despite the non-stationarity of the incoming data. Collectively, these methods illustrate how on-policy policy gradient may not provide the shortest path to an optimal policy.

In this paper we propose a novel approach for improving the policy optimization procedure. Rather than first defining an objective function and subsequently updating the policy parameters in the direction of its gradient, we directly define an update rule over the parameters. Critically, these update rules may not be the gradient of any objective function, mirroring an observation made in regards to the TD(0) algorithm [6]. Our contributions are the following:

1. We introduce an objective-free approach for constructing policy update rules,
2. establish convergence guarantees of these update rules, and
3. empirically demonstrate that these update rules are both sample efficient and stable under certain conditions.

We first present a family of policy updates that generalize the on-policy policy gradient along different components. Consequently, we name this family *generalized policy updates*. In the tabular setting, we provide sufficient conditions for these updates to converge to an optimal policy. We further illustrate and analyze the practical benefits of two update rules in this family on tabular domains and the Arcade Learning Environment [7].

2 Background

We consider the standard reinforcement learning (RL) framework in which an agent interacts with a Markov Decision Process (MDP) $\mathcal{M}$, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s_{t+1}|s_t, a_t)$ provides the transition dynamics, and $\mathcal{R}(s_t, a_t)$ is a reward function. A policy π defines a distribution over actions conditioned on the state, $\pi(a_t|s_t)$.

The value function of a policy π, computed at a state s, is the expected sum of discounted rewards obtained by following π when starting at state s:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t r(s_t, a_t) | s_0 = s \right] \tag{1}$$

The Q-value function of a policy at state s and action a is

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a \right] \tag{2}$$

Both value and Q-value functions are the fixed points of *Bellman operators* T^π. In particular,

$$(T^\pi Q)(s, a) = \mathcal{R}(s, a) + \gamma \sum_{\substack{s' \in \mathcal{S} \\ a' \in \mathcal{A}}} \left[\mathcal{P}(s'|s, a) \pi(a'|s') Q(s', a') \right] \tag{3}$$

We aim to find a policy π that maximizes the expected sum of discounted future rewards from an initial state distribution ρ with a discount factor $\gamma \in [0, 1)$. Given a policy π_θ parameterized by θ, the

canonical policy optimization objective can be written as follows:

$$J(\theta) = \sum_{s \in \mathcal{S}} \rho(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{4}$$

where $\pi_\theta(a|s)$ is the parameterized policy, and $Q^{\pi_\theta}(s, a)$ is the Q-value of the current policy. Computing $\nabla_\theta J(\theta)$ yields the on-policy policy gradient [31]:

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma} \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) \tag{5}$$

where the state distribution $d_\pi(s)$ is

$$d_\pi(s) = (1-\gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\rho, \pi_\theta), \tag{6}$$

corresponding to the discounted probability that the agent reaches state s when following policy π from start state $s_0 \sim \rho$.

In this work, we focus on *tabular softmax policies* with a discrete action space. We say a policy is tabular when it is represented with one parameter $\theta(s, a) \in \mathbb{R}$ per state-action pair. This is in contrast to policies that use function approximation, such as a neural network. We say it is a tabular softmax policy when these parameters are combined with a softmax function:

$$\pi_\theta(a|s) = \frac{e^{\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\theta(s,a')}} \tag{7}$$

3 Generalized Policy Update Rules

Actor-critic methods[4] simultaneously maintain a parametrized policy and a Q-value function. We consider an iterative procedure in which these are respectively denoted $\pi_{\theta_k} = \pi_k$ and Q_k at iteration k. The purpose of the Q-value function is to track this policy, which in turn is updated so as to favour high-value actions. The specific procedure we study here makes use of a step-size $\alpha > 0$, a parameter $m \in \mathbb{N}$, a state distribution $d_k(s)$, and an update direction $\Delta_k(\theta_k)$. It is described by the updates

$$\begin{aligned} &Q_{k+1} = (T^{\pi_{k+1}})^m Q_k, \\ &\theta_{k+1}(s, a) = \theta_k(s, a) + \alpha d_k(s) \Delta_k(\theta_k)(s, a). \end{aligned} \tag{8}$$

Our framework allows for variation in the state distribution d and the function Δ. The general form of Δ is

$$\Delta(\theta) = \sum_{a \in \mathcal{A}} d(a|s) g_\theta(s, a) M(s, a) \tag{9}$$

Within this definition, we highlight

1. A state-conditioned action distribution $d(a|s)$,
2. a gradient term $g_\theta(s, a)$, and
3. a gradient magnitude $M(s, a)$.

Critically, the on-policy policy gradient update (Equation 5) applied to tabular policies is an instance of this framework, since it sets $d(s) = d_\pi(s)$, $d(a|s) = \pi_\theta(a|s)$, $g_\theta(s, a) = \nabla_\theta \log \pi_\theta(a|s)$, and $M(s, a) = Q^{\pi_\theta}(s, a)$. Note that the omission of the sum over $\mathcal{S}$ arises specifically in the context of tabular policies, where changing the policy parameters of one state does not influence the policy at other states.

4 Conditions for Convergence

We now provide a set of conditions under which the iterative procedure described by Equation 8, in particular one that uses a generalized policy update rule, converges to an optimal policy. We begin with a few definitions.

[4]These methods include on-policy policy gradient if one considers Monte Carlo estimation as a form of critic.

Definition 1. *Let u, v be vectors in $\mathbb{R}^n$. We say that u is* order-preserving *w.r.t. v if*

$$v_i \geq v_j \implies u_i \geq u_j \; \forall \; 1 \leq i, j \leq n.$$

For two collections $\boldsymbol{u}, \boldsymbol{v}$ of $\mathcal{S}$-indexed vectors with $u(s), v(s) \in \mathbb{R}^n$ we say that $\boldsymbol{u}$ is order-preserving w.r.t. $\boldsymbol{v}$ if $v(s)$ is order-preserving w.r.t. $u(s)$ for each $s \in \mathcal{S}$.

We will find that a sufficient condition for convergence is for the updates Δ_k to be order-preserving with respect to the Q-values Q_k.

Lemma 1. *Let p_θ be a softmax distribution parametrized by $\theta \in \mathbb{R}^n$, i.e. $p_\theta \propto e^\theta$. Let $\boldsymbol{q}, \delta \in \mathbb{R}^n$, and suppose that δ is order-preserving w.r.t. $\boldsymbol{q}$. Then*

$$\sum_{i=1}^{d} p_{\theta+\delta}(i)\boldsymbol{q}(i) \geq \sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i).$$

Lemma 1 states that the expectation of a vector $\boldsymbol{q}$ under a softmax distribution is increased whenever its parameters are increased in an order-preserving direction. The proof of this lemma, and that of following results, can be found in the appendix.

Definition 2. *Consider the space of Q-value functions $\mathcal{Q}$ that map state-action pairs to a bounded interval in $\mathbb{R}$. Let $U : \mathcal{Q} \to \mathcal{Q}$. We say that the operator U is* action-gap preserving *if there exists a real value $c > 0$ such that*

$$\begin{aligned} Q(s, a_1) \geq Q(s, a_2) \implies \\ U(Q)(s, a_1) \geq U(Q)(s, a_2) + c(Q(s, a_1) - Q(s, a_2)). \end{aligned}$$

To prove convergence of the general iterative procedure, we view the update $\Delta(\theta)$ as an operator satisfying Definition 2. Our result will allow us to identify settings of $d(s)$, $d(a|s)$, and $M(s, a)$ that lead to convergent update rules.

Theorem 1. *Let d be a sequence of probability distributions over $\mathcal{S}$ with $d(s) > 0$ for all $s \in \mathcal{S}$. Let π_θ be a tabular softmax policy, and let $\alpha > 0$ be a step-size parameter. Consider the iterative procedure*

$$\begin{aligned} \theta_{k+1}(s, a) &:= \theta_k(s, a) + \alpha d(s) \Delta_k(\theta)(s, a), \\ Q_{k+1} &:= (T^{\pi_{k+1}})^m Q_k. \end{aligned}$$

Further suppose that $T^{\pi_{\theta_0}} Q_0 \geq Q_0$. Then provided $\Delta_k(\theta)$ is order-preserving w.r.t. Q_k for each k and the induced operator is action-gap preserving, the iterative procedure converges to an optimal policy:

$$\lim_{k \to \infty} Q^{\pi_{\theta_k}} = Q^*.$$

Theorem 1 is interesting for a number of reasons. First, it provides convergence guarantees for a large family of update rules, including natural actor-critic [25] and the Argmax Actor-Critic algorithm we describe below, itself related to MPO [1]. The proof technique, based on a monotonic improvement property derived from Lemma 1, provides further evidence that these policy-based methods can in fact behave similarly to value iteration or policy iteration methods, depending on the choice of α and m. We note that this result is a significant step from the typical convergence proof for policy gradient methods, which requires a two-timescale analysis. While noise in the value estimation process does impact performance, below we provide empirical evidence that either acting off-policy or using off-policy state distributions can increase the robustness of these updates to estimation noise.

The theorem also provides a means for deriving algorithms that are not the gradient of any objective function, by setting $d(s)$, $d(a|s)$, and $M(s, a)$ to alternatives that still satisfy the conditions. One simple example is to select these components to emulate value iteration.

We note that Theorem 1 does not include the usual on-policy policy gradient algorithm and the expected actor-critic algorithm presented below, whose updates are not in general order-preserving and are known to not ensure monotonic improvement [17]. Our proof also assumes a constant state distribution d, echoing negative results regarding policy iteration with non-uniform step sizes [8; 33]. Still, we believe our proof techniques may be extended to these algorithms with some additional care.

4.1 Expected Actor-Critic

We now consider a concrete instantiation of Equation 9, Expected Actor-Critic (EAC), which itself generalizes methods by Asadi et al. [4] and Ciosek and Whiteson [11]. We derive EAC by setting $d_k(a|s) = \pi_k(a|s)$ and $M_k(s,a) = Q_k(s,a)$, resulting in the following $\Delta_k(\theta)$:

$$\Delta_k(\theta) = \sum_{a \in \mathcal{A}} \pi_{\theta_k}(a|s) \nabla_{\theta_k} \log \pi_{\theta_k}(a|s) Q_k(s,a) \tag{10}$$

Note that EAC does not satisfy the sufficient conditions for convergence of generalized policy updates that we identify in Theorem 1. However, the EAC update rule with $d(s) = d_\pi$ is the standard policy gradient update, which is convergent. We study EAC since it falls within our generalized policy updates framework while connecting to standard actor-critic algorithms.

4.2 Argmax Actor-Critic

The second class of update rules we consider is Argmax Actor-Critic (AAC). We derive AAC by setting $d_k(a|s)$ to the uniform distribution $\frac{1}{|\mathcal{A}|}$ and $M_k(s,a) = \mathbb{1}_{a=\arg\max Q_k(s,\cdot)}$, resulting in the following value of $\Delta_k(\theta)$:

$$\Delta_k(\theta) = \sum_{a \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \nabla_{\theta_k} \log \pi_{\theta_k}(a|s) \mathbb{1}_{a=\arg\max Q_k(s,\cdot)} \tag{11}$$

Proposition 1. *The iterative procedure described by Equation 8, instantiated with the AAC update rule, converges to an optimal policy.*

5 Results

We empirically study the convergence speed and stability of the EAC and AAC updates to show that not all convergent updates are equal. We consider two simple MDPs that allow us to investigate the behavior of these update rules with no approximation error (Sections A.4 and A.5). We then derive practical, sample-based algorithms based on these update rules and demonstrate improved performance when compared to standard on-policy policy gradient and value-based RL in the Atari 2600 domain (Section 5.1).

5.1 Atari 2600 Experiments

To study generalized policy updates in a sample-based setting with function approximation, we develop practical algorithms based on EAC and AAC policy updates. We evaluate these algorithms on the Arcade Learning Environment (ALE; [7]) with sticky actions, using the recommended training protocol [22]. We modified the DQN architecture [24] by inserting a fully-connected layer at the penultimate layer of the network to output a stochastic policy. We provide details on the architecture and hyperparameters used in Appendix A.6.

The Q-function parameters, θ, are trained to minimize the evaluation Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t))^2\right] \tag{12}$$

with the target

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma \sum_{a' \in \mathcal{A}} \pi_{\bar{\phi}}(a'|s_{t+1}) Q_{\bar{\theta}}(s_{t+1},a') \tag{13}$$

where D is a replay buffer of previously sampled states and actions. The target in this loss function is a sample-based version of applying the Bellman backup operator T^{π_ϕ} to the current Q-function. Additionally, the update makes use of target Q-value and policy networks that are updated periodically.

The policy parameters, ϕ, for AAC are learned by minimizing the policy update loss $J_\pi(\phi) = -\mathbb{E}_{s_t\sim D}[\sum_{a\in\mathcal{A}} d(a|s_t) \log \pi_\phi(a|s_t) M(s_t,a)]$ where $d(a|s) = \frac{1}{|\mathcal{A}|}$ and $M(s,a) = \mathbb{1}_{a=\arg\max Q_\theta(s,\cdot)}$. For EAC, the following loss is minimized: $J_\pi(\phi) =$

	Mean	Median	> DQN
DQN	-8.3%	0.0%	0
On-Policy EAC	-1138.7%	-104.0%	4
EAC $+ \pi$	**1302.7**$\%$	14.3%	**40**
EAC $+ \epsilon$-arg max(π)	1123.6%	**14.7**$\%$	38
AAC $+ \pi$	626.3%	-6.1%	26
AAC $+ \epsilon$-arg max(π)	982.0%	8.2%	39

Table 1: Mean and median score improvement over DQN across 60 Atari games for each update rule and type of acting policy. Scores are normalized with respect to random and DQN agents (Equation 19). We also report the number of games for which each method outperforms DQN.

$-\mathbb{E}_{s_t \sim D}[\sum_{a \in \mathcal{A}} \pi_\phi(a|s_t) M(s_t, a)]$ where $M(s, a) = Q_\theta(s, a)$. Due to the use of a (large) replay buffer, the state distributions for these algorithms are off-policy.

In Table 1, we compare the performance of the EAC and AAC update rules across all 60 Atari games using 3 random seeds. We also compare their performance to DQN and on-policy EAC, which is equivalent to EAC but uses a small replay buffer so that the state distribution is on-policy. In addition to the update rule, we compare acting according to different strategies: acting directly according the policy (e.g. AAC $+ \pi$) or using epsilon-greedy action selection according to the argmax of the policy (e.g. AAC $+ \epsilon$-arg max(π)). The argmax of the policy is the action for which the probability is the highest. The EAC update rule in conjunction with acting on-policy (EAC $+ \pi$) achieves the best mean performance, whereas EAC $+ \epsilon$-arg max(π) achieves the best median performance.

6 Related Work

The EAC update rule has been previously studied in the policy gradient literature [4; 11]. However, the state distribution used is the on-policy discounted distribution, d_π, since the update rule is derived using the on-policy discounted expected reward objective. In our work, we show empirically that EAC performs well even when the state distribution is off-policy. The AAC update rule is closely related to probability matching [27] that minimizes the KL between the Boltzmann distribution induced by Q and a parameterized policy. AAC is also related to Soft Actor-Critic (SAC; [15]) as both minimize KL losses for the policy improvement step. While SAC minimizes the KL divergence between the policy π and softmax$(\frac{Q(s,\cdot)}{\tau})$ with a temperature parameter τ, AAC minimizes the reverse of this KL and drives τ to 0. The AAC update rule also relates to conservative policy iteration (CPI; [17]) and is a special case of the update rule introduced in Vieillard et al. [34] with $\alpha = 1$.

The policy learned via AAC can be viewed as a distillation of the argmax of the Q-function over training time, which is similar to policy distillation, a mechanism by which a teacher network can pass on knowledge to a student network [26]. One update considered in [26] is the cross entropy between a parametrized student policy and $a^* = \arg\max_a Q(s, a)$, which is precisely the AAC policy update rule. However, while [26] learn the Q-function using standard max Bellman targets, we find that using on-policy targets leads to better performance and is a principled approach with respect to the generalized policy update framework. Recent work in policy distillation shows that some distillation update rules have no corresponding objective [12], which is further support for the use of generalized policy updates.

7 Conclusion

This work aims to move beyond (surrogate) objective design by presenting a family of policy update rules obtained by varying components of the standard on-policy policy gradient. This resonates with approximate dynamic programming, where there is no obvious objective design, and is the basis of deep RL algorithms. Our goal is not to develop a new state-of-the-art algorithm but rather present an alternative, unified, perspective for performing policy optimization. Though there are multiple possibilities for selecting the different components of the update rule presented in Equation 9, we present a convergence result (Theorem 1) that can guide our design decisions.

References

[1] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *arXiv preprint arXiv:1907.04543*, 2019.

[3] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *Proceedings of the 36th International Conference on Machine Learning*, pages 151–160, 2019.

[4] Kavosh Asadi, Cameron Allen, Melrose Roderick, Abdel-rahman Mohamed, George Konidaris, Michael Littman, and Brown University Amazon. Mean actor critic. *stat*, 1050:1, 2017.

[5] Leemon C Baird. Reinforcement learning in continuous time: Advantage updating. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 4, pages 2448–2453. IEEE, 1994.

[6] Etienne Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 1993.

[7] Marc .G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[8] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[9] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[10] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

[11] Kamil Ciosek and Shimon Whiteson. Expected policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Wojciech M. Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *Proceedings of Machine Learning Research*, pages 1331–1340, 2019.

[13] Robert Dadashi, Adrien Ali Taiga, Nicolas Le Roux, Dale Schuurmans, and Marc G. Bellemare. The value function polytope in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1486–1495, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[14] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. In *Proceedings of the International Conference on Machine Learning*, 2012.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 2018.

[16] Ehsan Imani, Eric Graves, and Martha White. An off-policy policy gradient theorem using emphatic weightings. In *Advances in Neural Information Processing Systems*, pages 96–106, 2018.

[17] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[18] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[19] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of the International Conference on Machine Learning*, 2018.

[20] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 2000.

[21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.

[22] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[23] Erinc Merdivan, Sten Hanke, and Matthieu Geist. Modified actor-critics. *arXiv preprint arXiv:1907.01298*, 2019.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[25] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[26] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *International Conference on Learning Representations*, 2016.

[27] Philip N Sabes and Michael I Jordan. Reinforcement learning by probability matching. In *Advances in neural information processing systems*, pages 1080–1086, 1996.

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

[29] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[31] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[32] István Szita and András Lőrincz. Learning tetris using the noisy cross-entropy method. *Neural Computation*, 2006.

[33] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72, 2002.

[34] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Deep conservative policy iteration. *arXiv preprint arXiv:1906.09784*, 2019.

[35] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[36] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[37] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, figshare, 2010.

A Appendix

A.1 Proofs

Definition 3 (Action gap (Farahmand et al., 2011)). *Let $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a Q-value function. The action-gap $\mathcal{G}_Q(s)$ describes the difference between the largest and second largest elements of Q. If $a^* = \max_{a \in \mathcal{A}} Q(s, a)$, then*

$$\mathcal{G}_Q(s) = Q(s, a^*) - \max_{a \neq a^*} Q(s, a).$$

Lemma 1. *Let p_θ be a softmax distribution parametrized by $\theta \in \mathbb{R}^n$, i.e. $p_\theta \propto e^\theta$. Let $\boldsymbol{q}, \delta \in \mathbb{R}^n$, and suppose that δ is order-preserving w.r.t. $\boldsymbol{q}$. Then*

$$\sum_{i=1}^{d} p_{\theta+\delta}(i)\boldsymbol{q}(i) \geq \sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i).$$

Proof. We can rewrite the inequality in vector form:

$$p_{\theta+\delta}^T \boldsymbol{q} \geq p_\theta^T \boldsymbol{q}.$$

Let h be the scalar function such that:

$$h(\alpha) = p_{\theta+\alpha\delta}^T \boldsymbol{q}.$$

We will show that $h'(\alpha) \geq 0$ which will establish the claim. First, by the chain rule we have

$$\begin{aligned} h'(\alpha) &= \boldsymbol{q}^T \nabla_\theta(p_\theta) \nabla_\alpha(\alpha\delta) \\ &= \boldsymbol{q}^T (\mathrm{diag}(p_\theta) - p_\theta^T p_\theta)\delta \\ &= \Big(\sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i)\delta(i)\Big) \\ &\quad - \Big(\sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i)\Big)\Big(\sum_{i=1}^{d} p_\theta(i)\delta(i)\Big) \end{aligned}$$

Since δ is order preserving w.r.t $\boldsymbol{q}$, δ and $\boldsymbol{q}$ are equivalently sorted, we can thus apply the generalized form of Chebyshev's sum inequality to conclude

$$\Big(\sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i)\delta(i)\Big) - \Big(\sum_{i=1}^{d} p_\theta(i)\boldsymbol{q}(i)\Big)\Big(\sum_{i=1}^{d} p_\theta(i)\delta(i)\Big) \geq 0,$$

hence $h'(\alpha) \geq 0$. □

Theorem 1. *Let d be a sequence of probability distributions over $\mathcal{S}$ with $d(s) > 0$ for all $s \in \mathcal{S}$. Let π_θ be a tabular softmax policy, and let $\alpha > 0$ be a step-size parameter. Consider the iterative procedure*

$$\begin{aligned} &\theta_{k+1}(s, a) := \theta_k(s, a) + \alpha d(s)\Delta_k(\theta)(s, a), \\ &Q_{k+1} := (T^{\pi_{k+1}})^m Q_k. \end{aligned}$$

Further suppose that $T^{\pi_{\theta_0}} Q_0 \geq Q_0$. Then provided $\Delta_k(\theta)$ is order-preserving w.r.t. Q_k for each k and the induced operator is action-gap preserving, the iterative procedure converges to an optimal policy:

$$\lim_{k \to \infty} Q^{\pi_{\theta_k}} = Q^*.$$

Proof. We will show that the iterative procedure is convergent. Using Lemma 1, we have that if Δ_k is order preserving w.r.t. Q_k then

$$\pi_{\theta_{k+1}}(\cdot|s)^T Q_k(s, \cdot) \geq \pi_{\theta_k}(\cdot|s)^T Q_k(s, \cdot).$$

By assumption $T^{\pi_0}Q_0 \geq Q_0$, we can now use Theorem 1 in [23] to conclude that the iterative procedure is convergent.

We will now show that $Q_k \to Q^*$ for (3). Define Q as the limit of Q_k and suppose that Q has a single optimal action for all state (we can naturally extend to the case where there are multiple). We have that:

$$\exists K \in \mathbb{N} \text{ s.t. } \forall k \geq K, \mathcal{G}_{Q_k} \geq \frac{\mathcal{G}_Q}{2} = \epsilon.$$

This simply states that as the sequence $\{Q_k\}$ is approaching convergence (at some arbitrary iterate K), the action gap of the sequence is at least $\epsilon > 0$.

Now using our assumption, we have that

$$\exists c > 0, \forall k \geq K, \Delta_k(s, a^*) - \Delta_k(s, a) \geq c\epsilon.$$

Therefore,

$$\begin{aligned}
&\theta_{K+i}(s, a^*) - \theta_{K+i}(s, a) \\
&= \alpha d(s)\Big(\Delta_{K+i-1}(s, a^*) - \Delta_{K+i-1}(s, a)\Big) \\
&+ \Big(\theta_{K+i-1}(s, a^*) - \theta_{K+i-1}(s, a)\Big) \\
&= \alpha d(s) \sum_{j=1}^{i-1} \Big(\Delta_{K+j}(s, a^*) - \Delta_{K+j}(s, a)\Big) \\
&+ \Big(\theta_K(s, a^*) - \theta_K(s, a)\Big) \\
&\geq i\alpha d(s)c\epsilon + \Big(\theta_K(s, a^*) - \theta_K(s, a)\Big) \\
&\underset{i \to +\infty}{\to} \infty.
\end{aligned}$$

We can now conclude that $\{\pi_k\}$ converges to the greedy policy π_Q with respect to the limit of convergence Q. Therefore, as the Bellman operator is continuous we have:

$$Q = (T^{\pi_Q})^m Q.$$

Therefore, by the uniqueness of the fixed point of the Bellman operator, Q is the action value function associated with the policy π_Q which is greedy with respect to Q, hence $Q = Q^*$.

□

Proposition 1. *The iterative procedure described by Equation 8, instantiated with the AAC update rule, converges to an optimal policy.*

Proof. We first note that the $\nabla_\theta \log \pi_\theta(a|s)$ has the following form:

$$\frac{\partial}{\partial \theta(s, i)} \log \pi(a|s) = \begin{cases} 1 - \pi(i|s) & \text{if } i = a \\ -\pi(i|s) & \text{otherwise.} \end{cases} \tag{14}$$

This results in the following form of $\Delta_k(s, a)$:

$$\Delta_k(s, a) = \begin{cases} \frac{1}{|\mathcal{A}|}(1 - \pi(a|s)) & \text{if } a = \arg\max Q_k(s, \cdot) \\ \frac{-\pi(a|s)}{|\mathcal{A}|} & \text{otherwise.} \end{cases} \tag{15}$$

Therefore, $\Delta_k(s, \cdot)$ is order preserving and action-gap preserving with respect to Q_k. We can conclude using Theorem 1. □

Algorithm	$d(s)$	$g_\theta(s,a)$	$M(s,a)$
On-Policy AC	d_{π_θ}	$\nabla_\theta \log \pi_\theta$	Q^{π_θ}
On-Policy AC	d_{π_θ}	$\nabla_\theta \log \pi_\theta$	A^{π_θ}
Off-PAC	$d_\mu(s)$	$\nabla_\theta \log \pi_\theta$	Q^{π_θ}
ACE	$m(s)$	$\nabla_\theta \log \pi_\theta$	Q^{π_θ}
NPG	$d_{\pi_\theta}(s)$	$F(\theta)^{-1}\nabla_\theta \log \pi_\theta$	Q^{π_θ}

Table 2: Variation in the state distribution $d(s)$, gradient term, and gradient magnitude $M(s,a)$ for common policy update rules.

A.2 Relationship to Existing Methods

We show that our generalized policy updates framework unifies a number of existing policy update rules. These differ in how they define the three components $d(s)$, $g_\theta(s,a)$, and $M(s,a)$. Some common algorithms are instantiated in Table 2.

A.2.1 State Weighting

Changes in the state weighting dictate if the policy gradient is on-policy or off-policy. In particular, when the state distribution is the discounted state visitation distribution $d_\pi(s)$ of the current policy π_θ we recover on-policy policy update rules. In off-policy policy gradient algorithms, the state distribution is obtained from policies which are different from the policy being optimized.

Off-Policy Policy Gradient. The Off-Policy Actor-Critic (Off-PAC) algorithm [14] produces a gradient update that replaces $d_\pi(s)$ with the stationary state distribution $d_b(s)$ of a fixed behavior policy b. This update is obtained by changing the objective $J(\theta)$ to an off-policy objective $J_b(\theta) = \sum_s d_b(s) \sum_a \pi_\theta(a|s) Q^{\pi_\theta}(s,a)$, which optimizes the performance of the policy π_θ on states visited by b.

Off-Policy + Emphatic Weightings. Actor-Critic with Emphatic Weightings (ACE) replaces $d_b(s)$ with $d_b(s)i(s)$ in the off-policy objective above, where $i : \mathcal{S} \to [0, \infty)$ is an interest function. When computing the gradient of the modified objective, $d(s)$ in Equation 9 is set to be the emphatic weighting $m(s)$ [16].

In addition to the above methods, which use a fixed behavior policy to compute the state distribution $d(s)$, there are a number of actor-critic algorithms that sample states from a replay buffer. The replay buffer contains (s_t, a_t, r_t, s_{t+1}) transitions encountered by a mixture of past policies that were computed during the optimization process. Some notable examples are Actor-Critic with Experience Replay and Soft-Actor Critic [35; 15].

A.2.2 Magnitude

In the on-policy policy gradient (On-Policy AC), the magnitude function $Q^{\pi_\theta}(s,a)$ can be replaced by $Q^{\pi_\theta}(s,a) - b(s)$, where b is a state-dependent baseline function. The advantage function $A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$ is a frequently used instance of this [5; 29]. These settings of the magnitude function reduce the variance of the updates when estimating the gradient from samples while not biasing it.

A.2.3 Gradient Term

Natural policy gradient (NPG) and natural actor-critic algorithms [18; 25; 9] change the policy gradient by linearly transforming $\nabla_\theta \log \pi_\theta(a|s)$ using the inverse Fisher information matrix of the policy $F(\theta)^{-1}$. The motivation for this transformation is to make the gradient updates independent of the policy parameterization. Trust region policy optimization and proximal policy optimization constrain how much the policy changes in each update to achieve a similar result [28; 30] e.g. by imposing a KL constraint or by clipping the update.

Policy Update	$d(s)$	Time to Near-Convergence
EAC	d_π	1.0
EAC	$d_{\pi_{\text{uniform}}}$	0.29
EAC	$\frac{1}{\lvert\mathcal{S}\rvert}$	0.18
AAC	d_π	0.14
AAC	$d_{\pi_{\text{uniform}}}$	0.28
AAC	$\frac{1}{\lvert\mathcal{S}\rvert}$	0.14

Table 3: The time it takes EAC and AAC updates to obtain a near-optimal policy ($||V^\pi - V^{\pi^*}||_2 < 0.01$, where π^* is the optimal policy) on the two-state MDP for three different state distributions. The time to near-convergence is reported as the fraction of the time it takes the on-policy policy gradient to obtain a near-optimal policy.

A.3 Tabular Experiments

To demonstrate the flexibility in choosing components in our generalized policy updates framework, we consider three different choices for the state distributions $d(s)$. The on-policy discounted state distribution, d_π, is defined as in Equation 6. We also consider $d_{\pi_{\text{uniform}}}$, the discounted state distribution corresponding to using the uniform policy instead of the current policy in Equation 6. Finally, $\frac{1}{|\mathcal{S}|}$ is the uniform distribution over the state space $\mathcal{S}$.

A.4 Two-State MDP Experiments

To understand the changes in convergence speed of EAC and AAC update rules, we visualize the path of value functions induced by policies during the iterative procedure (Equation 8) on the value function polytope [13]. The value function polytope is obtained from the set of value functions of all possible policies, and it enables the visualization of locations on the polytope in which the algorithm (Equation 8) spends a large number of iterations with little change in the corresponding value function. We refer to these areas as *accumulation points*.

We recreate the setting from Dadashi et al. [13]: the MDP has two actions in each state and a uniform initial state distribution. The discount factor used is 0.9, and the transition and reward functions are described below. We use a tabular softmax policy (Equation 7) and the model of the environment to compute exact EAC and AAC policy updates. In each iteration k, we update the policy π_{θ_k} and then compute and plot its value function $V^{\pi_{\theta_k}}$ to overlay the path on the polytope. The learning rate is $\alpha = 1$.

We apply policy updates until the policy is near-optimal: the difference between the policy's value function and the value function of the optimal policy is less than 0.01. We visualize the accumulation points along the resulting value function paths and measure the number of updates required to reach near-convergence relative to EAC.

A.4.1 Value Function Path.

In Figure 1, we visualize the value function path taken by EAC and AAC updates with varying state distributions. EAC updates follow a path that traverses the edges of the polytope and visits corners other than the one corresponding to the optimal value function (Figure 1, top). The corners of the polytope correspond to value functions of policies that are nearly deterministic in both states, and they constitute a form of accumulation point [13]. EAC updates spend relatively more time in these areas, as indicated by the high concentration of value functions there (darker regions of the paths). By contrast, using AAC updates avoids these accumulation points and follows a more direct path to the optimal value function (Figure 1, bottom). Using off-policy state distributions ($d_{\pi_{\text{uniform}}}$ and $\frac{1}{|\mathcal{S}|}$) only marginally changes the shape of the path, but reduces the fraction of the time spent near or at the corners of the polytope.

A.4.2 Convergence Speed.

To verify that avoidance of accumulation points in the polytope yields faster convergence for AAC compared to EAC we measure the number of updates needed to obtain a near-optimal policy (see

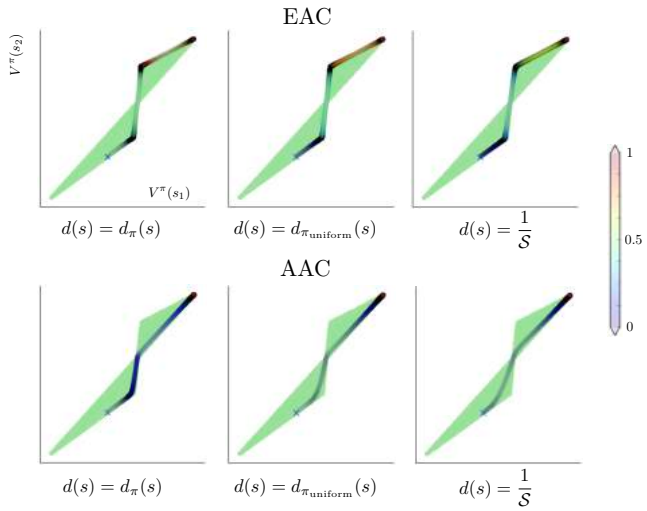


Figure 1: Value function paths generated by following EAC and AAC policy updates using three different state distributions on the two-state MDP. On the x-axis is the value function computed at state s_1, and on the y-axis is the value function computed at state s_2 for the policy π_k at each iteration k. The value function corresponding to the initial policy is indicated by a blue X. The darkness of each region along the policy optimization trajectory is proportional to the density of value functions in that region along the update path. The color of each point indicates the fraction of updates that have been completed (see the colorbar on the right).

Table 3). All variants of AAC require fewer steps compared to EAC, except when $d(s) = d_{\pi_{\text{uniform}}}$. In particular, AAC with $d(s) = d_{\pi}$ takes only 1/7th the number of updates as EAC to reach a near-optimal policy. With EAC updates, using off-policy distributions increases convergence speed.

A.4.3 Environment

The two-state MDP has $|\mathcal{S}| = 2$, $|\mathcal{A}| = 2$, and $\gamma = 0.9$. We use the following convention to describe the transition function $\mathcal{P}$ and reward function $\mathcal{R}$:

$$\mathcal{R}(s_i, a_j) = \hat{r}[i \times |\mathcal{A}| + j] \tag{16}$$

$$\mathcal{P}(s_k | s_i, a_j) = \hat{P}[i \times |\mathcal{A}| + j][k] \tag{17}$$

where $\hat{r} = [-0.45, -0.1, 0.5, 0.5]$ and $\hat{P} = [[0.7, 0.3], , [0.99, 0.01], [0.2, 0.8], [0.99, 0.01]]$.

A.4.4 Experimental Protocol

In the iterative procedure 8, we update the Q-value estimates Q_k in each iteration k by computing $Q_{k+1} = T^m Q_k$ with $m = \infty$. That is, $Q_k = Q^{\pi_k}$ for all k. We use this same procedure for the gridworld experiments.

Our motivation for using $m = \infty$ is so that the EAC update rule with $d(s) = d_{\pi}$ is equivalent to the on-policy policy gradient 5, which we aim to use as a point of comparison for analyzing convergence speed and stability when varying the type of update rule and state distribution.

A.5 Gridworld Experiments

To gain a better understanding of convergence speed, as well as the sensitivity of different update rules to random parameter initializations and Q-value estimation errors, we use a gridworld environment (Figure 2). This environment was previously studied in Ahmed et al. [3] to demonstrate the sensitivity of policy gradient algorithms to random restarts. As in the two-state MDP, the dynamics are known, which enables exact computation of the update rules in Equation 9.

The gridworld is of size 10×10 with two rewards: a reward of 5 and a distractor reward of 4.5 at the corners. The agent has four actions in each state. Here, we assume the agent always begins in the

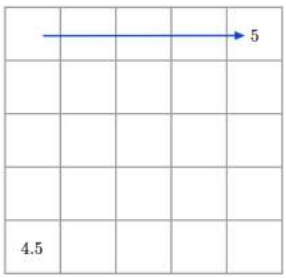
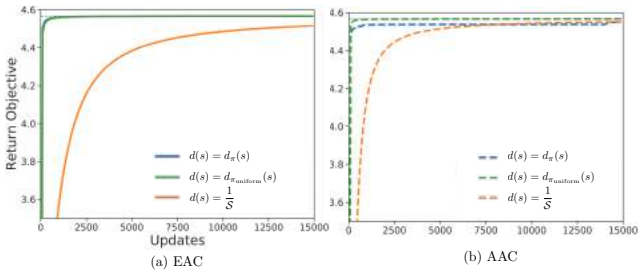


Figure 2: Gridworld environment. The agent starts in the upper left corner. The rewards are zero everywhere except for at the bottom left and upper right corners, which are terminal states. The optimal policy is illustrated with a blue arrow.



(a) EAC (b) AAC

Figure 3: Learning curves of EAC and AAC policy updates with varying state distributions $d(s)$.

upper left corner. From this initial state, the optimal policy is to go right, resulting in a discounted return of ≈ 4.56, where the discount factor is 0.99. We use 100 different initial values for the policy parameters and perform 15000 policy updates. We keep the set of initial values fixed across the different policy updates that we evaluate. As in the previous section, we optimize a tabular softmax policy using the iterative procedure (8) with learning rate $\alpha = 1.0$.

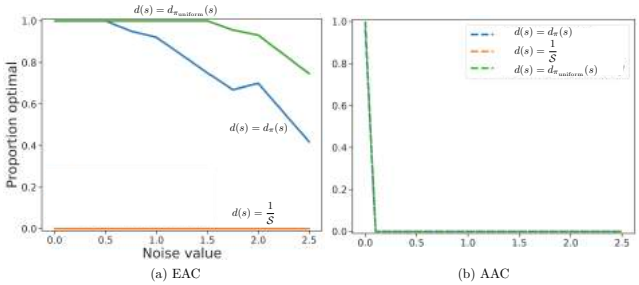
A.5.1 Convergence Speed.

Though Theorem 1 proves convergence for a class of policy updates that do not include EAC, we find empirically that EAC (where $d(s) \neq \frac{1}{|\mathcal{S}|}$) consistently quickly converges to the optimal solution for different parameter initializations[5] (Figure 3). While using the off-policy state distribution $d_{\pi_{\text{uniform}}}$ enjoys fast convergence, surprisingly, using a uniform state distribution, $\frac{1}{|\mathcal{S}|}$, slows convergence speed for both EAC and AAC.

A.5.2 Robustness to Value Function Noise.

In practice, value functions are not computed exactly and are instead estimated using sample-based TD updates via bootstrapping. To simulate this scenario, we add noise to the Q-values before calculating the update in Equation 9. Specifically, we add Gaussian noise with varying standard deviations to Q_k during each iteration, which affects the value of $M(s, a)$. We measure the effect of the noise on whether EAC and AAC updates converge to the optimal policy. Using $d(s) = d_{\pi_{\text{uniform}}}$ improves the robustness of EAC to noise in the Q-values (Figure 4a). Note that applying EAC updates with $d(s) = \frac{1}{|\mathcal{S}|}$ does not lead to convergence within 15000 updates, resulting in a flat curve. In contrast, AAC updates suffer when adding even a slight amount of value function noise. We expect that this phenomenon results from a change in the ordering of Q-values in the presence of noise, which results in an incorrect $\arg\max Q$ evaluation. To address AAC's sensitivity to this form of update estimation error, in the Atari experiments we consider changing the policy used to act in the environment.

[5]Unlike Ahmed et al. [3], we were unable to observe multiple local optima with our tabular policy parameterization.



(a) EAC (b) AAC

Figure 4: Proportion of initial parameter values that converge to an optimal policy in the presence of value function estimation error. On the x-axis, the noise value is the standard deviation used to produce Gaussian noise that is added to Q_k at each iteration k.

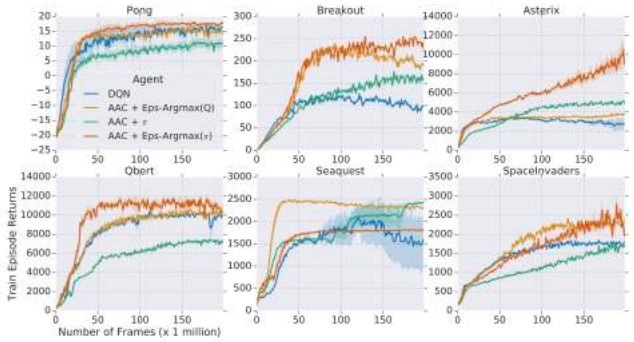


Figure 5: Performance of AAC with three different policies used to act in the environment: π, ϵ-$\arg\max(Q)$, ϵ-$\arg\max(\pi)$; compared to a DQN baseline.

A.6 Atari 2600 Experiments

A.6.1 AAC Ablation: Acting Policy

We examine the components used with the AAC policy update to determine which lead to good performance in practice. We specifically study the policy used to select actions in the environment; that is, the acting policy (ablations of additional components can be found in Appendix A.6.2). We perform ablations on six games: Pong, Breakout, Asterix, Qbert, Seaquest, and SpaceInvaders.

The AAC policy update takes a gradient step toward minimizing the KL divergence between the one-hot distribution indicating the action with the maximum Q-value and the policy. Specifically,

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D}[\text{KL}(\mathbb{1}_{a=\arg\max Q^{\pi_\theta}(s,\cdot)} || \pi_\phi(\cdot|s))] \tag{18}$$

If this KL loss were fully minimized at each step, then selecting actions with the policy would be equivalent to acting according to the argmax of the Q-function, which is similar to DQN's action selection. Therefore, we compare acting directly according the policy (AAC + π) with acting as in DQN; that is, we use epsilon-greedy action selection according to: (1) the argmax of the Q-values (AAC + ϵ-$\arg\max(Q)$) and (2) the argmax of the policy (AAC + ϵ-$\arg\max(\pi)$). Using AAC + ϵ-$\arg\max(Q)$ generally outperforms AAC + π (with the exception of Asterix and Seaquest), but AAC + ϵ-$\arg\max(\pi)$ most consistently achieves the best performance (see Figure 5). Since the policy constantly moves towards the argmax of the Q-function, acting according to the argmax of the policy for a particular state can be viewed as acting according to the action that most consistently maximizes the Q-function for that state. This can be seen as a mechanism for addressing AAC's lack of robustness to value function estimation error that was illustrated on the gridworld environment.

A.6.2 AAC Ablation: Critic Update

The target for the critic used in conjunction with AAC is a standard one-step on-policy TD target, as the Q-function is learned to approximate the expected discounted return under the policy π_θ. We experiment with bringing AAC closer to DQN by using $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a'} Q_{\bar{\theta}}(s_{t+1}, a')$.

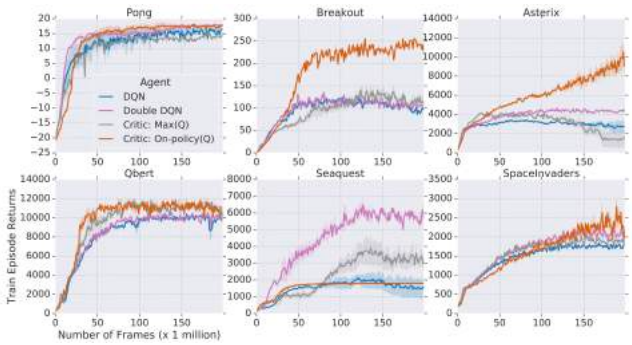


Figure 6: Two variants of AAC with differing critic updates: Max(Q) and On-policy(Q). The acting policy is ϵ-$\arg\max(\pi)$. We also include DQN and Double DQN as baselines.

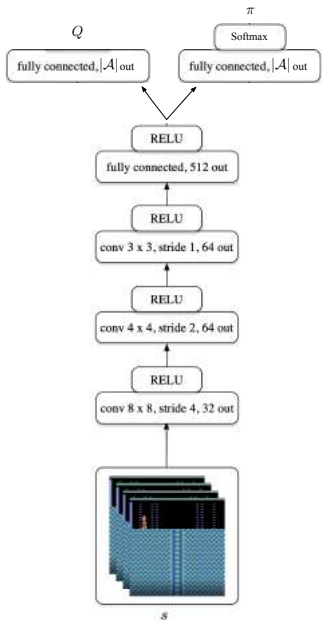


Figure 7: Network architecture used for all Atari experiments.

We find that this setting for the critic update performs on par with DQN or Double DQN on five games and is on-par with AAC on 4 games. However, AAC with the on-policy critic update most consistently achieves the best performance across the six games, indicating the importance of following the Q-function update procedure (8) (see Figure 6).

A.6.3 Network Architectures

The architecture used for the network outputting Q_θ and π_ϕ is illustrated in Figure 7. The policy network and Q value network are both single fully-connected layers on top of a shared base network.

A.6.4 Hyperparameters

The hyperparameters used for the Atari 2600 experiments are the same as the ones used for training DQN reported in Castro et al. [10]. The protocol we use to optimize the policy objective $J_\pi(\phi)$ is to use the Adam optimizer. To select the learning rate for each algorithm that contains a parameterized policy, we perform an ablation on 6 Atari games using learning rates for the Adam optimizer in the set $\{0.001, 0.0001, 0.00001, 0.000001\}$ and then select the learning rate that corresponds to the highest overall performance. We then use this hyperparameter setting to train the algorithm on all 60

Atari games (see Table 1). Finally, for on-policy EAC, we reduce the replay buffer size from 1000000 to 32 so that the states are sampled on-policy.

A.6.5 Score Normalization

To compute normalized scores, we follow the protocol from [2]. The improvement in normalized performance of an agent, expressed as a percentage, over a DQN agent is calculated as:

$$100 \times \frac{\text{Score}_{\text{Agent}} - \max(\text{Score}_{\text{DQN}}, \text{Score}_{\text{Random}})}{\max(\text{Score}_{\text{DQN}}, \text{Score}_{\text{Random}}) - \min(\text{Score}_{\text{DQN}}, \text{Score}_{\text{Random}})}. \tag{19}$$