Efficiently Managing Exploration and Gradient Stability in Deep Policy Gradient Methods

Vedant Nilabh, Richik Sinha Choudhury
Northeastern University, Boston, MA, USA
nilabh.v@northeastern.edu, sinhachoudhury.r@northeastern.edu

Abstract

We propose two augmentations to Deep Policy Gradient Methods with minimal computational overhead aimed at intelligently managing exploration and training stability. We evaluate the performance of these augmentations in common on-policy and off-policy PG methods, PPO and SAC, on standard environments.

Code, training statistics, and additional graphs for our paper can be found on our GitHub repository (https://github.com/richiksc/cs5180-final-project).

Introduction

Modern deep policy gradient (PG) methods, such as Proximal Policy Optimization (PPO) [Schulman et al.2017b] and Soft-Actor Critic (SAC) [Haarnoja et al.2018] have become the go-to for many Reinforcement Learning (RL) problems, especially given the ease of adapting to both discrete and continuous problems with minimal changes. One of the key motivators for these methods is that Vanilla Policy Gradient (such as REINFORCE, which uses Monte Carlo rewards, and Vanilla Actor Critic, which uses bootstrapping, to compute the policy gradient) these methods are high variance, even with TD, because you are approximating q with samples in the policy gradient update). A host of potential solutions beyond bootstrapping and baselines have been proposed to mitigate this, forming the basis of modern policy optimization methods, including trust region constraints, advantage estimation, soft updates, Polyak target averaging, and others. We wish to investigate if simple methods with minimal added computational overhead effectively stabilize learning while still making sure exploration is sufficient and tuned automatically. Our aim is to explore this trade-off between exploration and gradient stability in terms of context of modern policy optimization methods, and see if simple modifications can effectively tune this trade-off. Specifically we will explore the following two modifications to Discrete SAC and PPO:

1. Pre-filtering of gradients based on an EMA with an added residual of the current gradient based on Grokfast [Lee et al.2024].
2. Adding an additional term to entropy regularization, which is used in both PPO and SAC for exploration, conditioned on an EMA of the gradient norms, which we will refer to as Gradient Adaptive Entropy (GrAdE).

Background and Related Work

A number of existing approaches have been proposed to deal with high variance in policy gradients. Trust Region Policy Optimization [Schulman et al.2017a] uses the Natural Policy Gradient and includes a KL-divergence constraint in its objective, which is approximated with a second order Taylor series expansion and solved as a hard constraint on how the policy changes. However, TRPO is quite complex and hard to parallelize due to its use of conjugate gradients and line search to compute the updates. Other trust region constraint methods exist, including PPO. Another approach is off-policy PG methods. However, while these methods are sample efficient, they add even more variance, so learning effectively with them becomes an issue. Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al.2019] was one of the first deep off-policy PG methods to be proposed, making use of the deterministic policy gradient on deep networks to learn a deterministic policy which maximizes critic values, making it effectively a continuous action form of Deep Q Learning. DDPG also introduces Polyak averaging in the target networks for the critics, which adds stability. Soft Actor Critic (SAC) is another off policy PG method that extends DDPG, instead using a stochastic policy for continuous action spaces along with an entropy maximization objective added that makes the updates "soft."

Project Description

In this section we will formalize our MDP Structure on the envrionments upon which we tested as well as the algorithms we use in our experiments.

MDP Formalization

Four Rooms Environment The Four Rooms domain is formalized as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where:

$$\mathcal{S} = \{(x, y) \in Z^2 : 0 \leq x, y \leq 10\} \setminus \mathcal{W}$$
$$\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$$
$$\mathcal{P}(s'|s, a) = \begin{cases} 0.8 & \text{intended direction} \\ 0.1 & \text{first perpendicular direction} \\ 0.1 & \text{second perpendicular direction} \\ 1.0 & s' = s \text{ for wall collisions} \end{cases}$$
$$\mathcal{R}(s, a, s') = \begin{cases} 0 & s' = (10, 10) \\ -1 & \text{otherwise} \end{cases}$$

where $\mathcal{W}$ is the set of wall states, $s_0 = (0, 0)$ is the initial state, and episodes terminate at goal state $(10, 10)$ or after 459 steps.

Lunar Lander Environment The Lunar Lander domain is formalized as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where:

$$\mathcal{S} = \{(x, y, v_x, v_y, \theta, \omega, l_1, l_2) \in R^8\}$$
$$\mathcal{A} = \{0 : \text{do nothing}, 1 : \text{left engine},$$
$$2 : \text{main engine}, 3 : \text{right engine}\}$$
$$\mathcal{P} : \text{Box2D physics simulation with gravity} = -10.0$$

$\mathcal{R}(s, a, s') = r_t + r_e$ where:

$$r_t = \begin{cases} +10 & \text{per leg ground contact} \\ -0.03 & \text{side engine firing} \\ -0.3 & \text{main engine firing} \\ -|\theta| & \text{angle penalty} \\ -\|\mathbf{v}\| & \text{velocity penalty} \\ -|x| & \text{distance from pad} \end{cases}$$
$$r_e = \begin{cases} +100 & \text{safe landing} \\ -100 & \text{crash} \end{cases}$$

where episodes terminate upon crash, leaving viewport ($|x| > 1$), or successful landing. The initial state places the lander at the top center with a random initial force. An episode solves the task when total reward ≥ 200 and an agent solves the task if it can consistently solve over multiple episodes (typically 100).

Discrete Soft Actor-Critic

Discrete SAC [Christodoulou2019] modifies the original Soft Actor-Critic algorithm to handle discrete action spaces while maintaining the core idea of soft updates, maximizing both Values and Entropy. The key differences lie in the action space representation and subsequent simplified mathematical formulations:

Action Space and Network Outputs The Q-function and policy networks are modified to handle discrete actions:

- Continuous SAC: $Q : \mathcal{S} \times \mathcal{A} \to R$
- Discrete SAC: $Q : \mathcal{S} \to R^{|\mathcal{A}|}$

Instead of outputting Gaussian parameters, the policy network directly outputs a probability distribution, typically computed via the softmax function:

- Continuous SAC: $\pi_\phi(a|s)$ outputs (μ, σ) for Gaussian
- Discrete SAC: $\pi_\phi : \mathcal{S} \to [0, 1]^{|\mathcal{A}|}$ using softmax

Key Equations The state value function is computed directly without sampling:

$$V(s_t) = \pi(s_t)^T [Q(s_t) - \alpha \log(\pi(s_t))] \tag{1}$$

The Q-function objective remains similar but operates on discrete actions. It uses clipped double Q Learning to mitigate overestimation bias just like SAC:

$$J_Q(\theta) = E_{(s_t, a_t) \sim \mathcal{D}}[\frac{1}{2}(Q_\theta(s_t, a_t) - \tag{2}$$
$$(r(s_t, a_t) + \gamma E_{s_{t+1} \sim p(s_t, a_t)}[V_{\bar{\theta}}(s_{t+1})]))^2] \tag{3}$$

The policy improvement objective becomes:

$$J_\pi(\phi) = E_{s_t \sim \mathcal{D}}[\pi_t(s_t)^T [\alpha \log(\pi_\phi(s_t)) - Q_\theta(s_t)]] \tag{4}$$

Temperature Parameter The temperature objective for entropy tuning [Haarnoja et al.2019] is simplified to:

$$J(\alpha) = \pi_t(s_t)^T [-\alpha(\log(\pi_t(s_t)) + \bar{H})] \tag{5}$$

where $\bar{H}$ is the target entropy, typically set as some factor c times the maximal entropy for the discrete policy, which is $log(1/A)$ where A is the action dimension. Another importent detail is the polyak averaging in the target networks, which is adopted from DDPG, and adjusts the target critics as a linear interpolation from the current target and the current critic mediated by τ

$$\bar{\theta}_i \leftarrow \tau Q_i + (1 - \tau)\bar{\theta}_i \text{ for } i \in \{1, 2\} \tag{6}$$

The key difference is the fact that the discrete policy simplifies the objectives and allows for explicit computation, as opposed to having to use the reparametrization trick and approximate from samples due to the inherent added complexity of dealing with continuous policies.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy modern deep actor-critic method which builds upon the trust region constraint of TRPO, limiting policy optimization step size to avoid performance collapse. In comparison to TRPO, PPO maintains the trust region constraint by including a term in the objective function which penalizes large updates outside the function, instead of optimizing under a hard KL-divergence constraint. This enables the PPO objective function to be optimized using standard first-order optimization techniques based on stochastic gradient descent, i.e., Adam, making PPO much more computationally efficient than TRPO.

Variants There are two variants of PPO: **PPO-KL**, which penalizes the KL-divergence between policy action distributions in the objective function, and **PPO-Clip**, which discourages steps outside the trust region by limiting increases in the objective function when the policy probabilities π_θ diverge too far from the policy used to to collect training data π_{θ_k} in each PPO step. In this project, we focus on and implement PPO-Clip, since it is the most widely used deep policy optimization method due to its computational efficiency and good performance.

PPO-Clip Algorithm The objective function in PPO-Clip seeks to maximize advantage while constraining updates by using a *clipped surrogate advantage.* The advantages $\hat{A}_t$ can be estimated using any method of advantage estimation and use the critic network $V(s)$ as a baseline.

PPO-Clip computes the probability ratio between the current policy π_θ and the policy used to collect the current batch of training data π_{θ_k}:

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \tag{7}$$

The PPO-Clip objective function is defined as the following:

$$L^{\text{CLIP}}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{8}$$

The $\min$ function in the objective function ensures that "mistakes" made by the policy are sufficiently penalized without clipping a decrease in the objective function. If the policy decreases the probability of a "good" action with positive advantage to a ratio below $1-\epsilon$ or increases the probability of a "bad" action to a ratio above $1+\epsilon$, the unclipped surrogate advantage $r_t(\theta)\hat{A}_t$ is used instead.

The PPO-Clip objective is jointly optimized alongside a mean-squared error loss for the critic network based on returns and an entropy bonus to encourage exploration:

$$L(\theta) = L^{\text{CLIP}}(\theta) + L^{VF}(\theta) + H(\pi_\theta(s_t)) \tag{9}$$

Grokfast-EMA: Low-Pass Gradient Filtering

Grokfast-EMA modifies the gradients via a pre-filtering operation before the optimizer step by applying an exponential moving average (EMA) filter to amplify the low-frequency components of gradients. The algorithm treats the sequence of gradients as a stochastic signal and applies low-pass filtering. This is connected to the "grokking" phenomenon observed in deep networks in supervised learning [Power et al.2022], with the authors positing that amplifying the low frequency components can accelerate the rate of grokking. In the context of Reinforcement Learning, we posit that Grokfast can effectively stabilize training by reducing the noise via the low pass operation computed via the EMA, meaning after the initial exploration phase the policy can converge faster. The formalization for the algorithm is below:

Algorithm Description For a parameter θ updated via gradient descent, the update at timestep t is given by:

$$g_t = \nabla_\theta f(\theta_{t-1}) \tag{10}$$

where g_t is the gradient and f is the loss function. The EMA filter maintains a moving average μ_t of past gradients:

$$\mu_t = \alpha_g \mu_{t-1} + (1-\alpha_g) g_t \tag{11}$$

where $\alpha_g \in [0,1]$ is the momentum parameter controlling the filter's time constant. The modified gradient $\hat{g}_t$ amplifies low-frequency components by adding the filtered signal:

$$\hat{g}_t = g_t + \lambda \mu_t \tag{12}$$

where $\lambda > 0$ is an amplification factor for the low frequency component.

Algorithm 1 Grokfast-EMA

Parameters: scalar momentum α_g, factor λ
Input: initial parameters θ_0, objective function $f(\theta)$, optimizer's update function $u(g,t)$
begin: $t \leftarrow 0$, $\mu \leftarrow \mathbf{0}_{\dim\theta}$ ▷ Initialize EMA with zeros
while θ_t not converged **do**
  $t \leftarrow t+1$
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$ ▷ Calculate gradients
  $\mu \leftarrow \alpha_g \mu + (1-\alpha_g) g_t$ ▷ Update EMA
  $\hat{g}_t \leftarrow g_t + \lambda \mu$ ▷ Filter gradients
  $\hat{u}_t \leftarrow u(\hat{g}_t, t)$ ▷ Calculate update
  $\theta_t \leftarrow \theta_{t-1} + \hat{u}_t$ ▷ Update parameters
end while

GrAdE: Gradient Adaptive Entropy Regularization

Many policy-gradient algorithms use *entropy regularization* to encourage exploration. By including the entropy of action distribution in the objective function, the optimizer maximization encourages the action to select actions more randomly, increasing exploration. The simplest form of entropy regularization uses a constant entropy coefficient (κ) to compute an exploration bonus:

$$J(\theta) = J(\theta) + \kappa H(\pi_\theta(s))$$

Ideally, especially in on-policy methods, we would like to encourage and maximize exploration of the state-action space in the early stages of training, and then reduce exploration as training progress, enabling the converged policy to act greedily based on its learned parameters. This is often achieved through scheduling the entropy coefficient to follow a linear or exponential decay throughout the training session, sometimes with multiple stages. However, these schedules must be hand-developed and tuned based on the specific environment and RL problem, the RL algorithm in use, and the values of other hyperparameters.

In complex environments, evaluating different schedules for the entropy term can become compute- and time-intensive. We propose a simple method for intelligently tuning and scheduling entropy regularization. Our hypothesis is that gradient magnitudes are a good estimate of training progress, and keeping track of gradient norms will enable us to automatically scale the exploration bonus in response to convergence with minimal computational overhead.

GrAdE Algorithm The GrAdE algorithm [2] maintains an exponential moving average of the total Frobenius norm of the actor network gradients. This EMA is lazily initialized on the first policy update, preventing In addition, GrAdE also maintains a running maximum of this EMA, which it uses to compute a scaling factor r for the entropy bonus. GrAdE matches the entropy bonus with a constant coefficient κ when the EMA of gradient magnitudes is at its running maximum, near the beginning of training. If gradient magnitudes increase further, GrAdE will maintain this entropy bonus. As training progresses and gradient magnitudes decrease, r decreases, and GrAdE begins to decay the entropy bonus, effectively scheduling κ.

Algorithm 2 GrAdE Algorithm

Parameters: κ = entropy coefficient, β = smoothing factor
Input: θ = policy parameters
begin: $\mu \leftarrow \sum ||\nabla_\theta J(\theta)||$ ▷ Initialize EMA
begin: $\mu_{max} \leftarrow \mu$
for each policy update **do**
  $\theta \leftarrow \theta_t$ ▷ Latest policy parameters
  $g_t \leftarrow \sum ||\nabla_\theta J(\theta)||$ ▷ Compute gradients
  $\mu \leftarrow \beta\mu - (1-\beta)g_t$ ▷ Update EMA
  $\mu_{max} \leftarrow \max(\mu_{max}, \mu)$
  $r \leftarrow \mu/\mu_{max}$ ▷ Compute scaling factor
  $J(\theta) \leftarrow J(\theta) + \kappa r * H(\pi_\theta)$ ▷ Apply entropy bonus
end for

Experiments

Discrete SAC

We use Discrete SAC and adopt it to the Four Rooms Environment and Lunar Lander. We use a replay buffer to sample transitions and do one update for each step we take in the environment in all experiments. Importantly, all hyperparameters will be listed below, many of which are the same as the discrete SAC paper. Note all networks are 3 layers for Discrete SAC (one hidden layer). Four Rooms results are for 1000 episodes averaged over 5 trials and Lunar Lander Results are for 350,000 steps averaged over 3 trials. Note that α_g is the smoothing parameter in Grokfast-EMA while α is the temperature parameter in Discrete SAC. Also Note that Four Rooms uses 1 update for 1 environment step, while Lunar Lander does 50 environment steps and then 50 updates. Rewards are also clipped in Lunar Lander if they fall below -10. SAC is the most widely used off policy deep PG method, making it a good testbed for our modifications. Below, we list our hyperparameters. Note that we test all 4 combinations (Discrete SAC, Discrete SAC with only Grokfast-EMA, Discrete SAC with only GrAdE, and with both) on four rooms but only test vanilla Discrete SAC, and Discrete SAC with grokfast on LunarLander, due to computational constraints. Note that in evalaution, we follow convention from [Christodoulou2019] use the arg-max action for consistency in evaluation. The Networks follow the initialization scheme used by DDPG.

Table 1: Hyperparameters for Discrete SAC in Four Rooms

Hyperparameter	Value
Optimizer	Adam
Learning rate	3×10^{-4}
Discount factor (γ)	0.99
Target network update rate (τ)	0.005
Replay buffer size	1×10^4
Batch size	256
Hidden layer	[256]
Activation function	ReLU
Initial temperature (α)	1.0
Target entropy	$-\log(1/\lvert\mathcal{A}\rvert) \cdot 0.5$
Initial random steps	5,000

Table 2: Hyperparameters for Discrete SAC in LunarLander

Hyperparameter	Value
Optimizer	Adam
Learning rate	3×10^{-4}
Discount factor (γ)	0.99
Target network update rate (τ)	0.005
Replay buffer size	1×10^6
Batch size	256
Hidden layer	[256]
Activation function	ReLU
Initial temperature (α)	0.2
Target entropy	$-\log(1/\lvert\mathcal{A}\rvert) \cdot 0.6$
Initial random steps	10,000

Discrete SAC results For Four Rooms, GrAde made almost no difference and actually made learning worse, as seen by the learning curves. We think this is because our discrete SAC implementation already incorporates an adaptive entropy tuning mechanism, so entropy is adjusted based on this already. Further, the idea of annealing the entropy bonus in the loss is less relevant for SAC due to it being off policy: in most cases, having some entropy bonus is actually seen as a good thing. We also incorporated GrAdE only in the actor, and used the existing alpha times an EMA of the gradient norms times the entropy: it really wasn't adding anything on top of the entropy tuning, where we had already selected a good entropy target that already provided a good trade off between exploration and exploitation for Four Rooms. Therefore, we didn't choose to include the loss plots or gradient magnitudes for GrAdE, as they look pretty similar to their non-GrAdE counterpart.

However, in the case of both Lunar Lander and Four Rooms, Grokfast-EMA actually is able to converge slightly faster while being more stable. In the case of Four Rooms, the training curve is higher initially while stabilizing to a comparable value at the end. Further, when looking at critic losses, we can see that they stabilize earlier in the case of Grokfast, which helps explain the better learning, as the actor can only learn from the critic values, so if the critic converges faster, the actor will follow, hence Grokfast helping.

Table 3: Hyperparameters for Grokfast-EMA and GrAdE

Hyperparameter	**Value**
Grokfast EMA Smoothing parameter (α_g)	0.98
Amplification parameter (λ)	2.0
Entropy coefficient (κ)	α
GrAdE EMA Smoothing Factor (β)	0.9

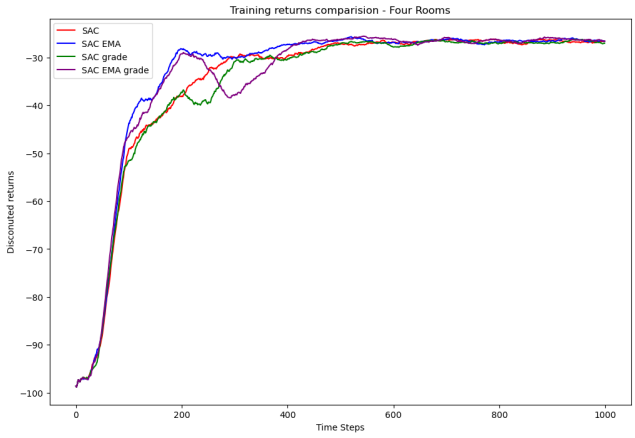

Figure 1: Four Rooms Learning Curves, all configurations

This is also reflected in the gradient magnitudes are reduced by a factor of 5 and 10 in the actor and critic respectively with Grokfast, indicating significant noise reduction. This trend continues in Lunar Lander where we see in both training returns and evaluation returns that Grokfast is able to converge faster. One interesting thing to note is Grokfast actually does worse in both the training and evaluation curves initially but the quickly makes up ground and exceeds the baseline discrete SAC. We think this could be because lunar lander is a noisier environment, so it takes time for the EMA to be an effective low pass filter, but when enough gradients are accumulated the Grokfast agent is able to converge rather quickly relative to the baseline. This is also reflected in the critic losses and gradients, where we see that critic dramatically decreases at 100000 time steps as do the critic gradient magnitudes, while the baseline shows only modest declines here. Further, the gradient magnitudes are even more significantly reduced than four rooms, as the critic gradients are further away even on a log scale, and the actor magnitudes are also significantly reduced. This behavior actually serves to indicate that Grokfast actually might be better in noisier environments due to how it is able to dynamically adjust to noisy gradients and converge more efficiently.

PPO

Being the most widely used Deep PG method, our PPO-Clip implementation served as a testbed for our proposed augmentations in an on-policy setting. We implemented PPO-Clip using full Monte-Carlo returns in our advantage computation:

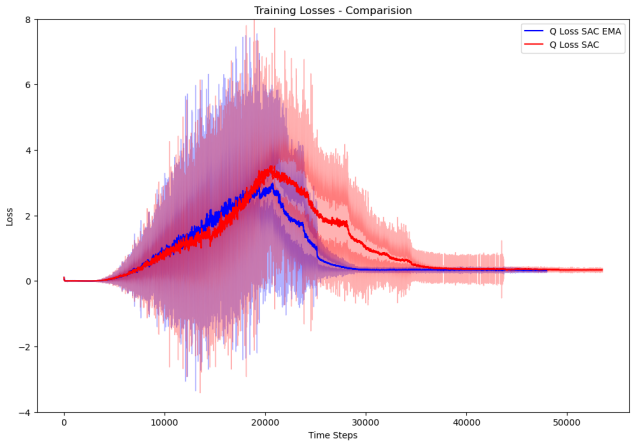

Figure 2: Critic losses for Four Rooms with Confidence Bands

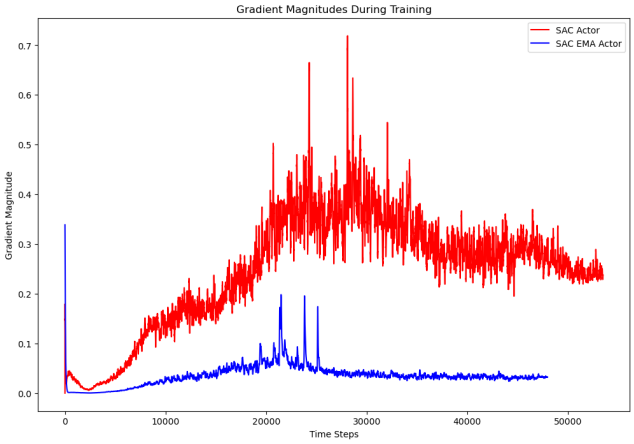

Figure 3: Actor Gradient Magnitudes for Four Rooms

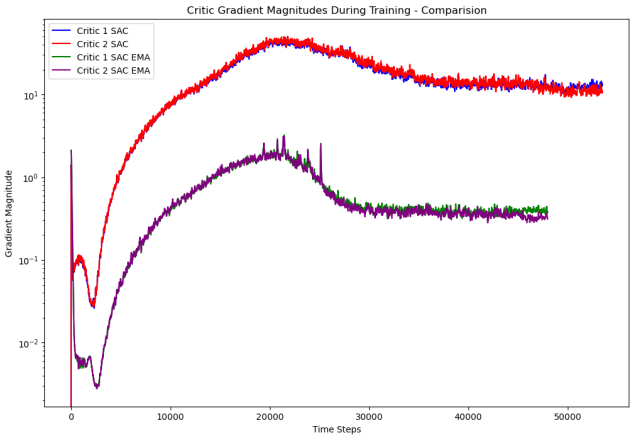

Figure 4: Critic Gradient Magnitudes for Four Rooms, Log Scale

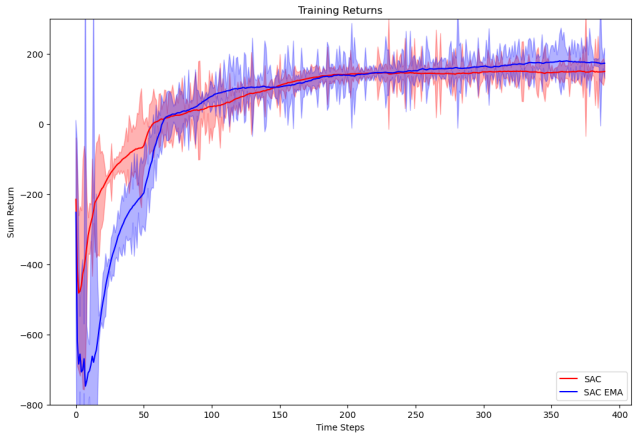

Figure 5: Training Curves for Lunar Lander - SAC and SAC with Grokfast-EMA

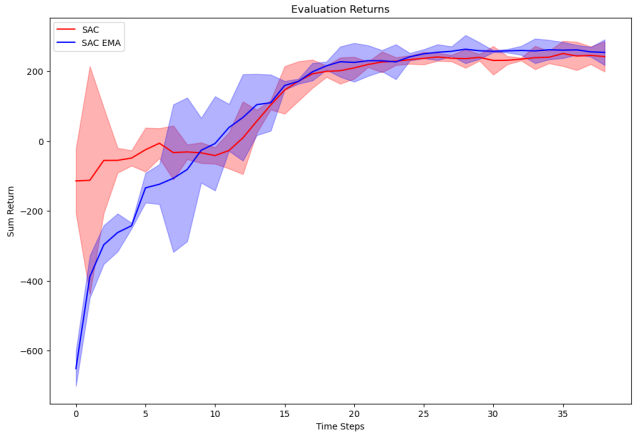

Figure 6: Evaluation Curves for Lunar Lander - rollouts every 10 episodes

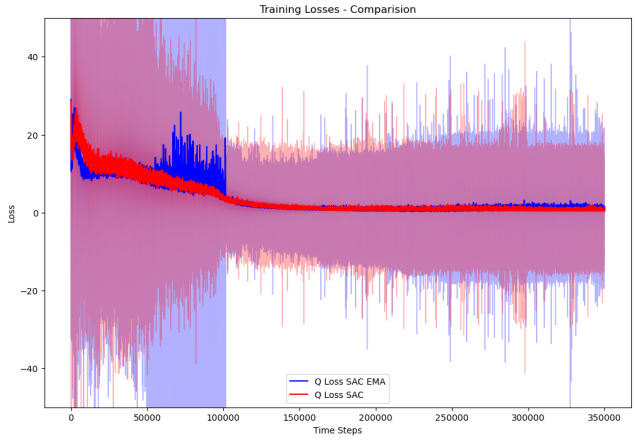

Figure 7: Critic Losses for Lunar Lander with Confidence Bands

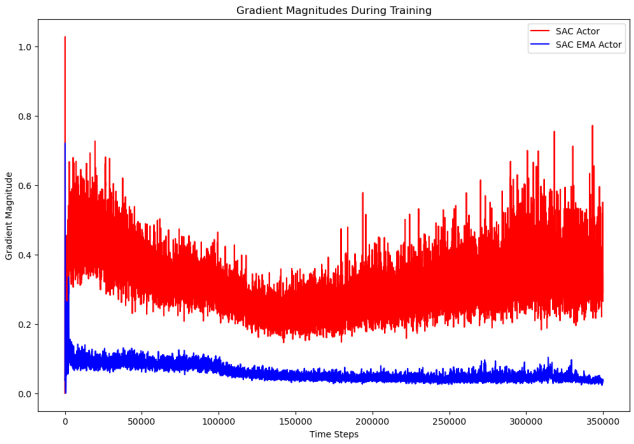

Figure 8: Actor Gradient Magnitudes for Lunar Lander

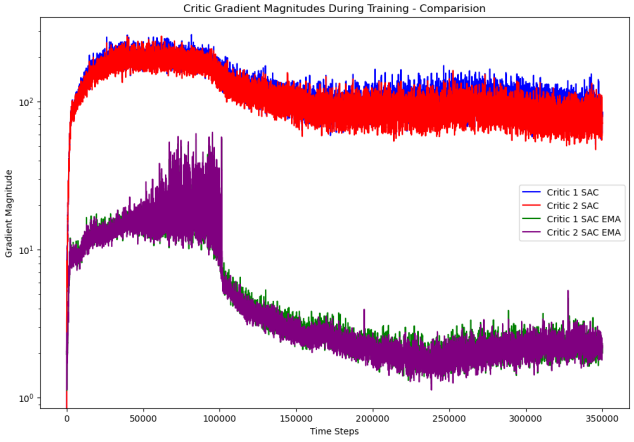

Figure 9: Critic Gradient Magnitudes for Lunar Lander

$$\hat{A}_t = G_t - V(s_t) \tag{13}$$

In addition, we incorporated return and advantage normalization to maintain stability.

We conducted four experiments with our PPO implementation, training an agent for 5 trials each on the Lunar Lander environment from Gymnasium:

1. Vanilla PPO-Clip with no modifications as a baseline
2. PPO with Grokfast
3. PPO with GrAdE
4. PPO with GrAdE and Grokfast

In the Four Rooms environment, we were unable to get our PPO implementation to learn, regardless of hyperparameter values and training batch sizes. Hence, no experiments were conducted in Four Rooms. Because PPO is an on-policy method, it fails to learn in environments with sparse reward signals (such as Four Rooms), in comparison to off-policy methods, which are less sensitive to current trajectories. PPO requires trajectories with positive reward signals to be sampled for adequate learning, which is very unlikely early in training in the Four Rooms environment. In addition, previously collected trajectories have to be discarded once PPO updates are made due to the trust region constraint, so the policy quickly loses any successful trajectories collected.

Hyperparameters for Lunar Lander

PPO Hyperparameters:

Hyperparameter	Value
Number of episodes	1500
Training batch	1 episode
Num. epochs	5
Adam learning rate	5×10^{-4}
Discount factor (γ)	0.99
Clipping parameter (ϵ)	0.2
Entropy coefficient κ	0.01
Hidden layers	[128, 128]
Activation function	ReLU

GrAdE Hyperparameters:

Hyperparameter	Value
Entropy coefficient κ	0.02
EMA smoothing factor β	0.9

Grokfast Hyperparameters:

Hyperparameter	Value
EMA smoothing factor α	0.98
Amplification parameter λ	2.0

PPO Results

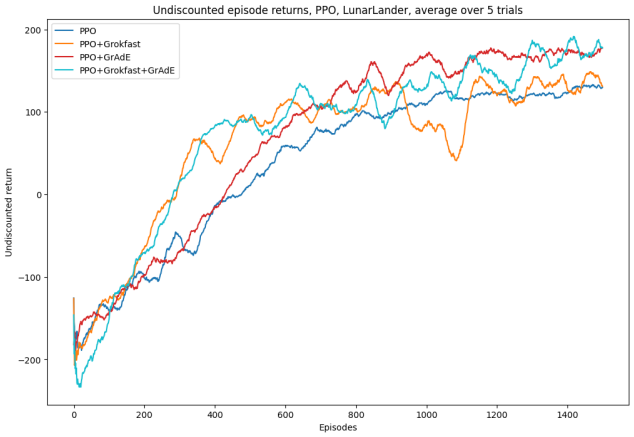

Figure 10: Lunar Lander learning curves, PPO, all experiments

GrAdE Results When comparing total rewards per episode (undiscounted return), the standard measure of performance in the Lunar Lander environment [Figure 10], we find that the addition of GrAdE scheduling results in noticeably better performance at convergence, receiving about 50 more reward than its vanilla PPO counterpart. In our GrAdE experiment, we increased the entropy coefficient to 0.02 compared to 0.01 in the vanilla PPO setting to take advantage of scheduling and encourage more initial exploration. However, we also noticed a similar, but smaller, improvement in episode returns at convergence when we ran GrAdE with $\kappa = 0.01$, illustrating that GrAdE effectively schedules the exploration bonus in response to training performance, enabling the policy to act more greedily at convergence. This becomes clearly apparent when comparing the value of the entropy bonus in the objective function $\kappa r * H(\pi_\theta)$ during each training experiment [Figure 11], showing an effective decay in response to training progress. When we look at the policy action distribution entropy itself [Figure 12], we find that the smaller entropy bonus term at convergence is not just in response to a decayed scaling factor, but that the policy entropy itself has reduced in response, resulting in increased asymptotic performance.

Grokfast Results When comparing the training performance of our PPO with Grokfast experiments to their counterparts without Grokfast [Figure 10], we find that although both improve more quickly in the early stages of training, they are less stable at convergence with similar average performance to their counterparts. This contradicts our initial hypothesis that gradient filtering would be an effective method to encourage training stability at convergence. We posit that this is because the trust region constraint in PPO, in addition to return and advantage normalization is already effective at reducing the variance of updates, therefore, the existing gradients are a low-noise, low-frequency signal. Adding the amplified EMA of the gradients with Grokfast to the existing gradients simply amplifies the overall gradients, making it equivalent to increasing the learn-

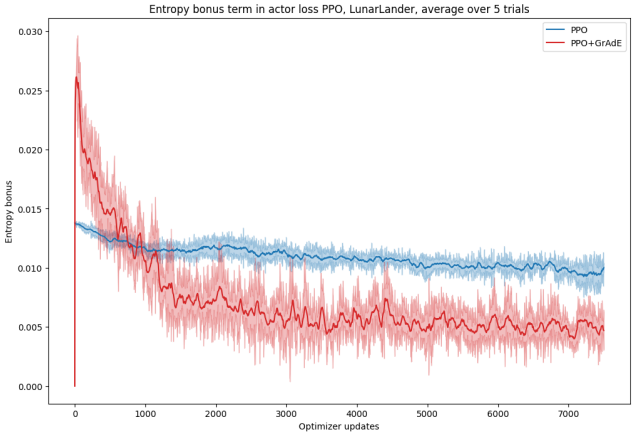

Figure 11: Entropy bonus term, PPO Lunar Lander

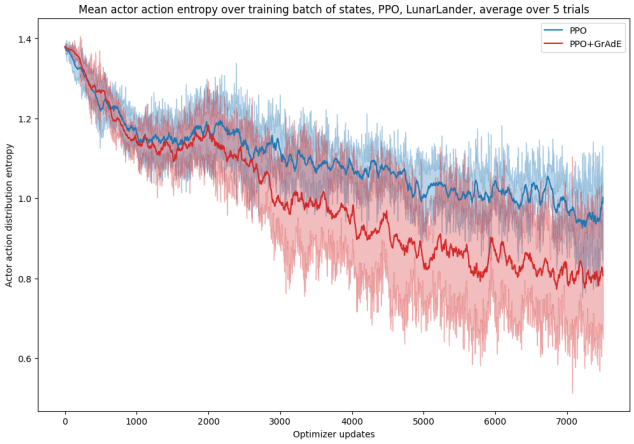

Figure 12: Policy entropy, PPO Lunar Lander

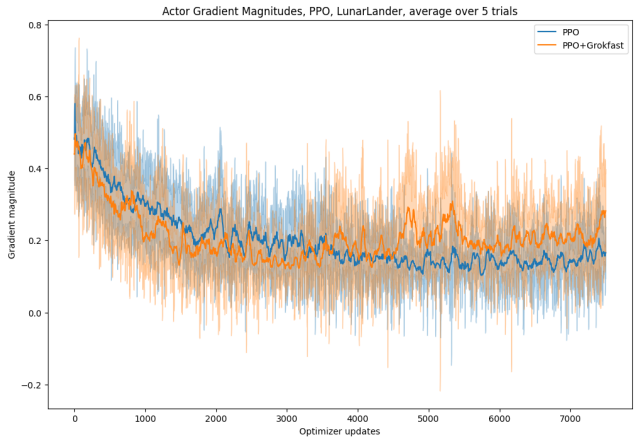

Figure 13: Actor Gradient Magnitudes

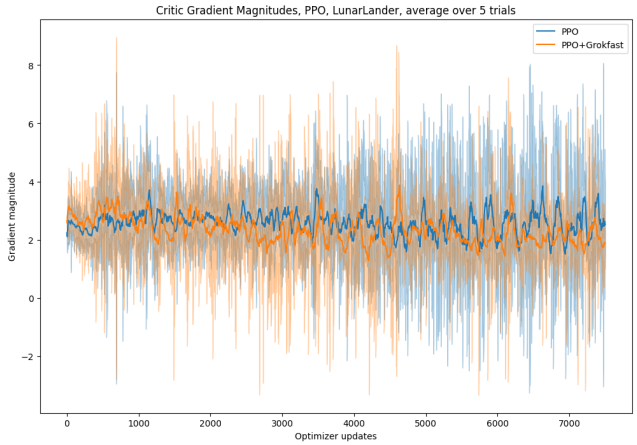

Figure 14: Critic Gradient Magnitudes

ing rate. With our amplification parameter of $\lambda = 2$, this would be equivalent to tripling the learning rate if the gradients were a noiseless signal. This results in more aggressive but unstable learning, explaining the pattern in the learning curves.

We can confirm this by examining the gradient magnitudes of the actor and critic over the course of the training experiments [Figures 13 and 14].

Conclusions

After evaluating our two augmentations on two widely-used Deep PG methods, PPO and SAC, we find that each of the two augmentations are better suited to a certain type of algorithm. GrAdE (Gradient Adaptive Entropy Regularization) is more useful in on-policy algorithms, which require sufficient exploration early and are evaluated with the same policy used to train. GrAdE does not help and may even hurt SAC with Entropy Tuning, which already has an adaptive mechanism to adjust the entropy bonus to achieve a target model entropy. However, a fairer comparison may be applying GrAdE to SAC without Entropy Tuning.

We also found that Grokfast is better suited to off-policy methods like SAC, or other PG methods with noisier gradients, resulting in an order-of-magnitude reduction in gradient norms and faster actor and critic learning. This also brings to mind the deadly triad of RL, where off-policy methods with function approximation and bootstrapping can suffer from instability, so Grokfast being useful here, even with the existing stabilization measure that SAC uses, gradients can still be quite noisy. However, we learned that trust region methods like PPO, especially in conjunction with existing gradient stability methods like advantage and return normalization, are already highly effective at maintaining gradient stability. Since the gradients are already a low-noise signal, therefore, applying gradient filtering with Grokfast can actually harm stability and policy performance. Importantly, we think the fact that Grokfast isn't as useful for PPO, along with our graphs, serves as a good indicator of our PPO implementation actually being extremely stable.

References

[Christodoulou2019] Christodoulou, P. 2019. Soft actor-critic for discrete action settings.

[Haarnoja et al.2018] Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

[Haarnoja et al.2019] Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; and Levine, S. 2019. Soft actor-critic algorithms and applications.

[Lee et al.2024] Lee, J.; Kang, B. G.; Kim, K.; and Lee, K. M. 2024. Grokfast: Accelerated grokking by amplifying slow gradients.

[Lillicrap et al.2019] Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2019. Continuous control with deep reinforcement learning.

[Power et al.2022] Power, A.; Burda, Y.; Edwards, H.; Babuschkin, I.; and Misra, V. 2022. Grokking: Generalization beyond overfitting on small algorithmic datasets.

[Schulman et al.2017a] Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2017a. Trust region policy optimization.

[Schulman et al.2017b] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017b. Proximal policy optimization algorithms.