REGULARIZING NEURAL NETWORKS FOR EFFICIENT SOFT-ACTOR CRITIC

Thanakul Wattanawong, Kevin Mo
Department of Computer Science
University of California, Berkeley
{j.wat,kevmo}@berkeley.edu

EXTENDED ABSTRACT

We are studying techniques that regularize neural networks used to approximate Q-functions in a high update-to-data ratio such as ReDQ (Chen et al. (2021)) and DroQ (Hiraoka et al. (2021)). These methods offer superior sample efficiency compared to vanilla implementations of their underying algorithms (e.g. Soft Actor Critic). We examine overall how well these methods generalize through an extensive series of experiments that aim to isolate the effects of each change.

Our first section motivates and runs experiments over four Mujoco environments that were also studied in Hiraoka et al. (2021). We conduct a hyperparameter sweep over classical neural network regularization methods such as L_1 and L_2 regularization, and show that these methods improve over SAC on some environments, mostly on Ant and Humanoid. We emphasize the need to choose hyperparameters carefully per environment as we saw a 4x difference in the performance improvement at $O(10^5)$ depending on the choice of λ, and even performance drops in some cases. We also study the use of spectral normalization and find that it can perform well. One benefit of this method is there are no hyperparameters to tune and it generally improves reward overall.

Next, we reimplement ReDQ and DroQ on top of a Soft Actor Critic implementation (Bin Li) that learns to drive a car in the CARLA (CARLA Team). Smith et al. (2022) had found that the relative ranking of algorithms can depend on the task and so we wanted to explore and see how well the aforementioned methods would perform on a different task. Furthermore, data collection in CARLA can require long GPU simulation if realism is needed, and we make use of the parallel data collection environment to see how the reward scales with reduced data collection rates.

Our results on CARLA indicated that ReDQ generally performs the best and that DroQ performs similarly to vanilla SAC. Further experimentation shows that ReDQ is very computationally expensive, taking 3x the wall time to run compared to SAC. We halve the ensemble size to try and counter this effect and reduce wall time by 40% with a small performance drop. We also attempt to adjust the dropout rate of DroQ to improve its performance without success. Finally, we adjust the data collection rate by varying the number of parallel agents available and find that the regularized methods (DroQ, ReDQ) can generally still perform well while SAC cannot learn properly at high update to data ratios.

We draw several conclusions from our experiments. The first is that regularization is critical in the high update-to-data regime, and it is essential in several environments to learn anything at all. Next, we argue that sample efficiency can come at a high compute cost, but also remark that sample collection can often be parallelized and so the absolute wall time will depend on the type of computational resources available. Finally, based on these two insights, we propose that it is possible to vary the update-to-data ratio either in an offline or online manner in order to learn as efficiently as possible. We run some initial experiments on CARLA to show that a high update-to-data ratio performs exceptionally well in the early stages but does not perform well in the long run, suggesting that this ratio may benefit from optimization.

Our code and results are available at https://github.com/ja5087/sac_carla_exploration.

1 INTRODUCTION

Deep Reinforcement Learning has shown much promise for solving complex tasks including games (Silver et al. (2016)), robotics control tasks (Kober et al. (2013)), and even Text-to-SQL (Zhong et al. (2017)). However, efficiency still remains a concern, as many RL algorithms generally require a large number of samples to be successful. Collecting samples is a resource consuming operation, requiring lots of compute or memory in the for simulations or real-life rollouts.

In this paper, we study variants of Soft-Actor Critic (Haarnoja et al. (2018)), a model-free off-policy deep reinforcement learning algorithm. Off-policy methods can be more sample efficient as they can learn from previously collected samples that used a different policy for data collection. Soft Actor Critic itself is fairly popular and is known to be robust across many hyperparameter values.

Many methods have been proposed to improve the sample efficiency of SAC. For example, one may increase the the update-to-data ratio, i.e. the number of gradient updates taken by the critic per data collection run. To alleviate the instability caused in this setting, algorithmic and architectural improvements such as ReDQ (Chen et al. (2021)) and DroQ (Hiraoka et al. (2021)) have been proposed.

These methods may be seen as regularization techniques that help avoid Q-function overfitting. However, it still remains unclear as to how well these methods generalize across other tasks and what specifically motivates their superior performance. Existing literature and our own experiments show that performance on these algorithms can vary widely between environments and hyperparameters. Existing applications of RL in real-life robotics often try a variety of techniques to discover what works well on their task (Smith et al. (2022)).

Furthermore, when discussing efficiency it is imperative to thoroughly consider all aspects of computing. Hiraoka et al. (2021) found that ReDQ's sample efficiency came at the cost of memory, compute and wall time, all of which were several times higher than SAC and DroQ. Although it is often true that sample collection often dominates runtime, expensive architectural changes such as ReDQ can challenge this assumption.

In this work, our contributions are threefold. First, we motivate and explore additional regularization techniques to expand the search space of possible architectural changes. Second, we implement these changes in a parallel Soft-Actor Critic implementation that learns to drive in the CARLA self-driving car simulator to further investigate how well these methods generalize. Finally, having established what methods work well in a high update-to-data setting, we turn our attention to proposing some ways to tackle the trade off between data collection, a highly parallelizable task, to critic update, which is a highly synchronous task.

In general, we find that methods such as L1 and L2 regression can also perform well on many of the Mujoco environments. On the CARLA car driving simulator, we find that ReDQ continues to offer superior performance compared to SAC, SAC with high update-to-data ratio, and DroQ. We explore additional changes to improve the efficiency and performance of these methods through extensive ablations and profiling experiments.

2 RELATED WORK

Soft-Actor Critic is an off-policy actor-critic algorithm that adds an entropy term while optimizing the policy in order to explore a wider scope of near-optimal behaviors Haarnoja et al. (2018). Specifically, the policy term is:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t))]$$

Although maximizing entropy can detract from maximizing reward, experiments show that Soft Actor Critic is generally superior across many tasks in addition to being more stable across multiple seeds.

ReDQ is an algorithm that aims to achieve higher sample efficiency by increasing the update-to-data ratio and stabilizing it using an ensemble of Q functions of which a random subset are minimized during eICLRach update step Chen et al. (2021). Specifically, a key contribution they make is highlighting the importance of the standard deviation of the Q-function bias, defined as:

$$Z_{M,N} = \gamma(\max_{a' \in \mathbb{A}} \min_{j \in \mathbb{M}} Q^j(s', a') - max_{a' \in \mathbb{A}} Q^\pi(s', a'))$$

This bias equation captures the difference between the target critic and the ground truth Q-value as measured empirically with rollouts from the policy. ReDQ further derives relationships between the ensemble size, minimization subset size, and the bias term. Increasing the ensemble size reduces the variance of the bias term, and increasing the subset size lowers the mean of the bias, as there is a minimization over a maximization of Q functions.

DroQ DroQ proposes adding Dropout and LayerNorm on top of Q functions, and the authors have shown that this leads to higher returns with high sample efficiency without incurring the cost of an ensemble Hiraoka et al. (2021). The dropout values are tuned as hyperparameters per environment. However, many results from the DroQ paper indicate, for example, that while ReDQ is generally superior on hard tasks such as hard tasks such as Walker2d, Ant, and Humanoid, it is actually worse than vanilla SAC for Hopper. Additionally, with a higher ensemble size, it is actually more beneficial for sample efficiency to only use LayerNorm. In general, while the use of Dropout and LayerNorm can have synergistic effects in certain scenarios, it is not yet well understood how neural net architectures can help optimize learning.

Sample Efficient Model-Free RL Work on measuring sample efficiency and finding new ways to improve it has been ongoing. For the three algorithms we mention specifically, Smith et al. (2022) has compared them on a real world robotics task of walking and generally found that while DroQ achieves the best performance, it is on par with only SAC + LayerNorm with an update to data ratio of 20. In general, the generalizability of architectural improvements to Q-functions is not well understood, and the authors suggest that it is the use of regularization and normalization itself that yields the most benefits, not any one particular method. They also highlight ReDQ's relatively high computational requirements compared to other methods.

Accelerated Reinforcement Learning It is clear that in many cases, sample collection is a highly parallelizable task. Authors such as Stooke & Abbeel (2018) and Rudin et al. (2021) have shown that despite the possibility of many challenges inherent to distributed systems such as stragglers, distributed and parallel RL methods are fast and feasible up to thousands of environments.

3 Exploring Additional Regularizers in Mujoco Environments

3.1 Experimental Setup

We base our code off Hiraoka et al. (2021)'s GitHub repository[1]. On top of re-running the experiments in Figure 2 of the paper for SAC, ReDQ and DroQ, we implement additional regularization techniques and compare them to the SAC baseline. Due to a limit on computational resources, we were only able to run the experiments once per experiment. Finally, in Appendix A we summarize all our results.

The parameters used to run the experiments are encoded as JSON files in our GitHub repository.

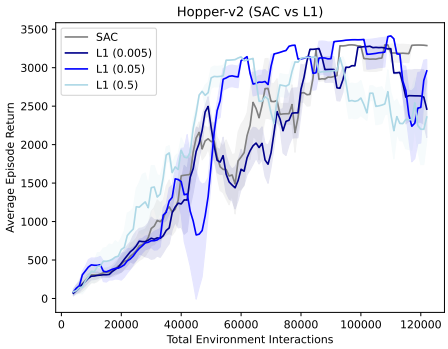
3.2 L1 Regularization

We explore L1 Regularization on the weights of the Q-functions. Specifically, our loss function for training the critic is given by the following equation:

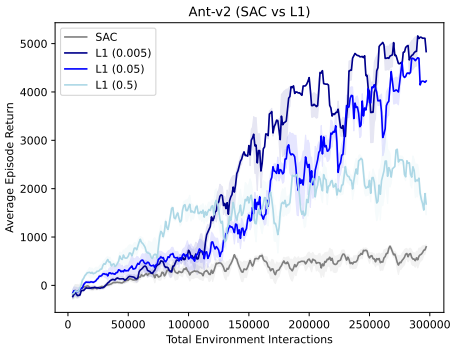
$$L = (y - \hat{y})^2 + \lambda_{L1} \sum |w_i|$$

[1] https://github.com/TakuyaHiraoka/Dropout-Q-Functions-for-Doubly-Efficient-Reinforcement-Learning

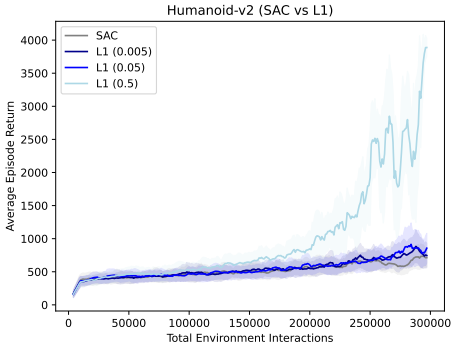
Adding the L1 norm loss term trains the weights to be sparse (Google), and we hypothesize that it will encourage the model to filter out noise in the rollouts and prevent overfitting. We present our results below:



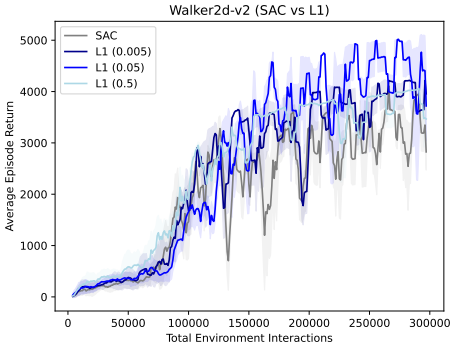
(a) Hopper-v2



(b) Ant-v2



(c) Humanoid-v2



(d) Walker2d-v2

Figure 1: L1 regularization with $\lambda_{L1} = 0.5, 0.05, 0.005$ compared with SAC on a collection of experiments. The x-axis represents the number of environmental interactions and the y-axis represents the average reward, with the error bar representing the standard deviation of the reward during each evaluation period.

As seen in Figure 1, on easy environments such as Hopper-v2, all methods perform similarly. However, for environments such as Ant and Humanoid, L1 regularization can offer significant advantages. Choosing λ_{L1} remains an important task, as there is a 2.5x difference in reward at 3×10^5 environment steps on Ant-v2 between the most optimal and least optimal λ_{L1} values.

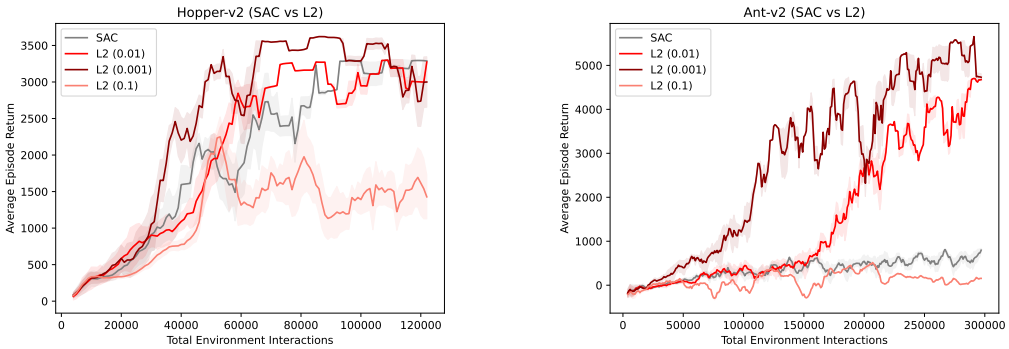
3.3 L2 Regularization

In a similar vein, L2 regularization penalizes parameters with a large magnitude. It does this by adding a loss term that scales with the square of the parameters, resulting in the following loss function:

$$L = (y - \hat{y})^2 + \lambda_{L2} \sum |w_i|$$

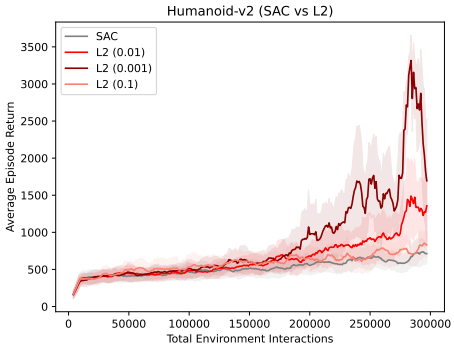
While L1 loss tends to drive parameters to zero, L2 merely scales them to be closer to zero. We present our results below over a variety of λ_{L2} values.

Our results are much more varied on L2 as evidenced by Figure 2. In particular, $\lambda_{L2} = 0.1$ causes a catastrophic drop in performance compared to the SAC baseline on Hopper, and suboptimal results on Ant v2. For Ant specifically, smaller L2 values perform very well, and $\lambda_{L2} = 0.001$ offers a 10x increase in performance over our SAC baseline. and we suspect that L2 values that are too

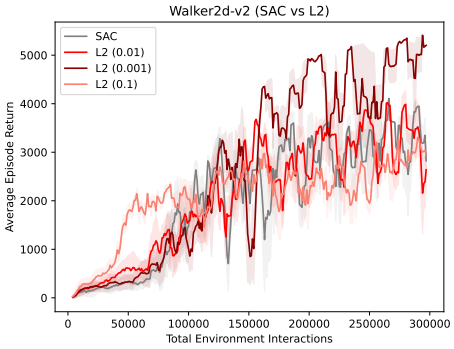


(a) Hopper-v2

(b) Ant-v2



(c) Humanoid-v2



(d) Walker2d-v2

Figure 2: L2 regularization with $\lambda_{L2} = 0.1, 0.01, 0.001$ compared with SAC on a collection of experiments.

large cause the model to learn too little. Furthermore, while performance improves on Humanoid-v2 somewhat there is some instability as evidenced by the sharp drop at around $s = 3 \times 10^5$ for $\lambda_{L2} = 0.001$.

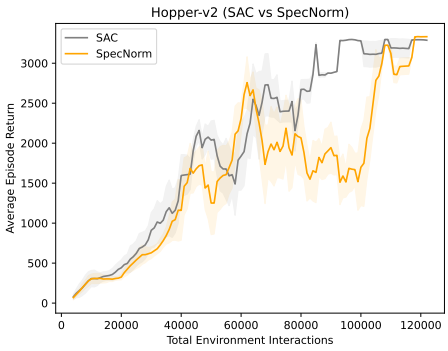
3.4 Spectral Normalization

Spectral Normalization is a well-known technique for regularization in the GAN literature (Miyato et al. (2018)). Gogianu et al. (2021) showed results that indicate that it may be also be useful in a reinforcement learning context. In particular, they claim that it generally does not cause performance degradation and so is useful in most situations. We test out this assumption by adding spectral normalization on the second-last layer of the Q-function as was the case in the paper.

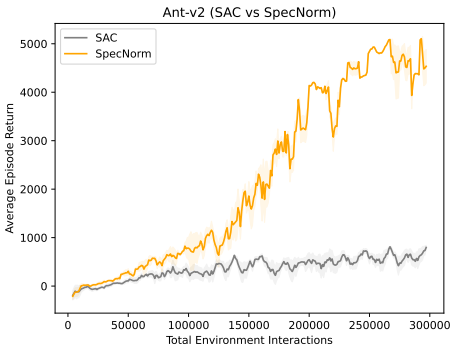
In general, Spectral Normalization is also useful overall, except on Hopper-v2 where it caused some instability around $s = 10^5$. This further reinforces the idea that there are many regularization techniques that work well. Spectral Normalization is a pretty simple change and has mostly beneficial or no effect on reward at least in our considered environments.

3.5 Comparison of All Methods + ReDQ + DroQ

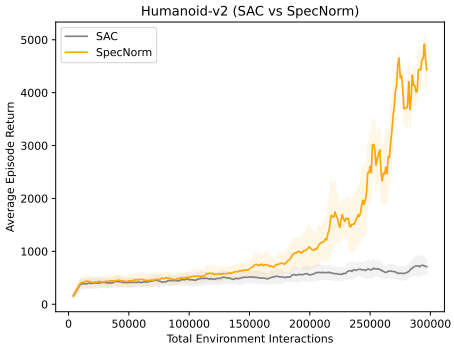
We have also reproduced the experiments that use ReDQ and DroQ based on those in Hiraoka et al. (2021), and present them in Appendix A. We do not extensively analyze them here as the overall performance of all regularization methods can be pretty similar with the appropriate amount of tuning, suggesting that there may be many other regularization methods that can benefit Soft Actor Critic with high update to data ratio. Our results suggest a need to both conduct more extensive architecture searches and attain a greater understanding of why these methods work so well for



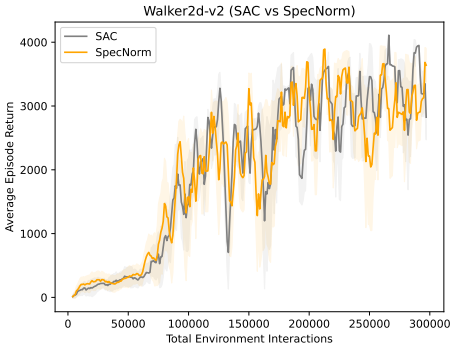
(a) Hopper-v2



(b) Ant-v2



(c) Humanoid-v2



(d) Walker2d-v2

Figure 3: Spectral normalization compared with SAC on a collection of experiments.

Q-function regularization. We draw limited conclusions from our data as we were unable to run repeated trials due to compute limitations, but our scripts are open-source and available.

4 Experimenting on the CARLA Simulator

Having established that many regularization/normalization methods are feasible for SAC in the high update-to-data setting, we implement these algorithms on a more challenging task in the CARLA autonomous driving simulator (CARLA Team) in order to further explore how well these algorithms perform under various scenarios.

4.1 Experimental Setup

We base our implementation off an existing implementation of parallel Soft Actor Critic in Bin Li and reimplement ReDQ and DroQ in the PaddlePaddle codebase. Since the CARLA simulator is GPU-accelerated, for all experiments we allocate one GPU per environment, and an additional GPU to train the neural networks in order to avoid contention. All experiments were run on a server with a 48-core Intel(R) Xeon(R) Gold 6126 CPU and 8 RTX Titans.

The authors started with 3 parallel agents ($a = 3$) that return in aggregate 3 environment rollouts per policy/critic update, and so we use that as our baseline and call it UTD 1 for simplicity's sake. The parameters used to run the experiments are available on our GitHub repository.

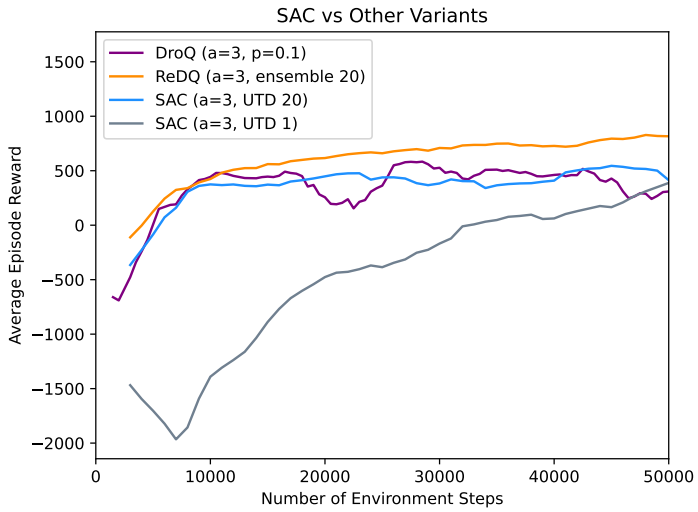


Figure 4: Comparison of the performance of all algorithms in CARLA

4.2 CARLA Results

4.2.1 Comparison of Algorithms

Figure 4 illustrates that ReDQ continues to outperform DroQ and SAC on this task. In particular, SAC with UTD 1 performed the worst in the first 50k steps. What is surprising is that simply increasing the update to data ratio worked well enough for SAC.

4.2.2 Making ReDQ more efficient

We noticed that ReDQ took about three times as much as SAC to reach 50k steps (15h vs 5h). Therefore, we decided to lower the ensemble size to see how it would affect performance as shown in Figure 5. It turns out this degraded performance slightly but decreased runtime to about 9h (+40%), which may be an acceptable tradeoff.

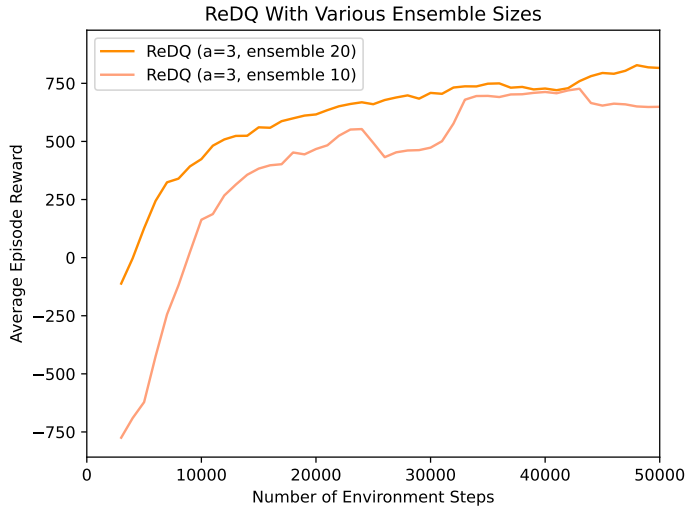


Figure 5: Comparison of ReDQ ensemble sizes in CARLA

4.2.3 IMPROVING THE PERFORMANCE OF DROQ

DroQ did not improve much on this benchmark compared to vanilla SAC at UTD 20. We decided to investigate if varying the dropout rate would change anything. Figure 6 indicates that changing the dropout rate to 0.01 does not have much of an effect on the performance. In general, Hiraoka et al. (2021) have said that tuning the dropout rate optimally is still an open question.

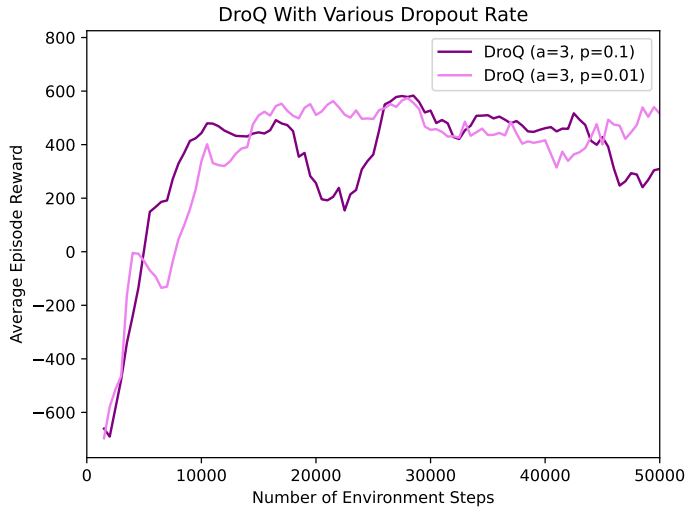


Figure 6: Comparison of DroQ dropout rate in CARLA

4.2.4 DECREASING DATA PER UPDATE STEP

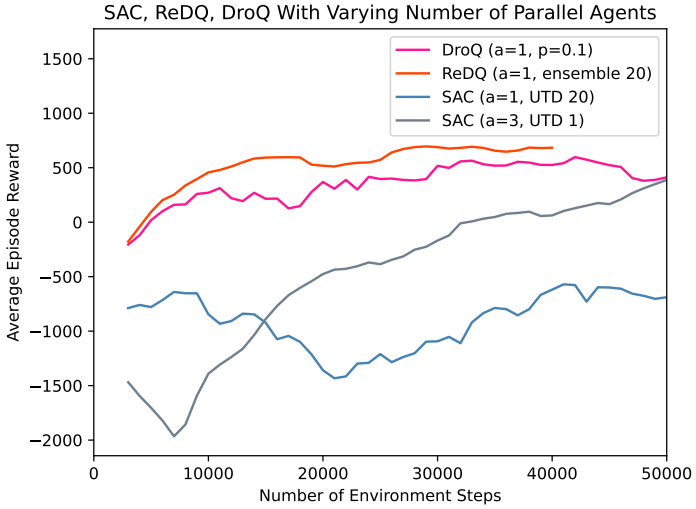


Figure 7: Adjusting data collection rate in CARLA

Finally, we took advantage of the parallel RL implementation to decrease the amount of rollouts per update step by decreasing the number of parallel agents. Note that each step on the x-axis represents an environment interaction. We decreased the number of parallel agents to 1, and kept the update to data ratio at 20. Results from figure 7 this case we see clearly that SAC with 20 updates fares exceptionally badly while other algorithms retain their performance, proving the necessity of regularization methods in the high update-to-data regime.

5 Lessons learned about regularized SAC variants

5.1 Regularization is essential in the high update-to-data regime

Our experiments in section 3 indicate that regularization techniques like ReDQ and DroQ, and even seemingly simple changes such as L1 and L2 regularization can improve average episode reward significantly. In fact, for environments such as Humanoid and Ant, regularization is almost essential to learn anything at all.

5.2 Sample Efficiency Can Come at High Compute Cost

So far, we have considered sample-efficient training, which is to reach the highest performance given a fixed number of environment samples. Reinforcement learning is usually dominated by sample collection time due to the cost of collecting rollouts. However, with a high update to data ratio or ensembled algorithms, one may start to see critic and actor updates take up more time than expected.

However, it is well known that sample collection can be highly parallelized given a fixed policy and numerous episodes (Rudin et al. (2021)). On the other hand, synchronous critic and policy updates are a serial task. If we are able to parallelize the sample collection and assume ideal job packing, we may be able to efficiently trade off computing resources for less wall time.

5.3 Efficient Tradeoff Between Data Collection and Learning Algorithm Cost

Given that we are able to learn at a high update-to-data ratio, it may even be possible to optimize not just the data collection, but the rate itself jointly. Our experiments in section 7 showed that regularization can help assist learning at high update-to-data ratios. We are of course free to vary that ratio and reduce the cost of updating the critic and agents in order to find the optimal ratio to learn in the least amount of wall time given a limited amount of computing resources.

We propose that it may be worth studying methods to optimize the update-and-data collection rates offline, or optimize the ratio online during learning. For example, policies in the beginning may require more updates to learn a reasonable policy, but may need to collect relatively more data later on in the process to improve beyond a certain point.

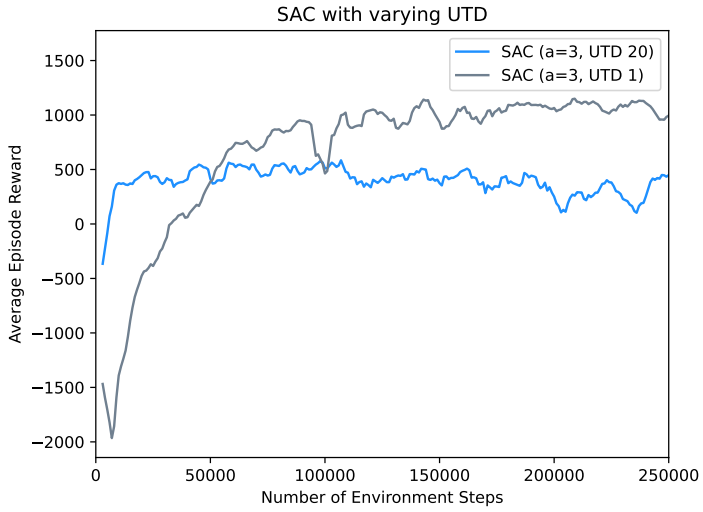


Figure 8: Adjusting only update-to-data ratio in CARLA

We study the long-run learning behavior in CARLA for SAC of UTD 1 and 20 in Figure 8 and demonstrate that while high UTD performs well in the beginning, the UTD 1 agent ultimately reaches a higher reward by $s = 2.5 \times 10^5$. This lends credence to our idea that a dynamic update-and-data schedule may benefit reinforcement learning algorithms.

6 GROUP CONTRIBUTIONS

Thanakul Wattanawong led the initial proposal of the original project and headed research into Soft-Actor Critic and related work. In addition to contributing original analysis of experiment results, he also led the exploration into the CARLA simulator and extended the existing implementations of Soft Actor Critic to implement additional regularization algorithms.

Kevin Mo led the coordination of the experiment runs and result visualizations, contributing code related to interpreting experiment results. In addition to assisting with the initial proposal planning, he performed research review and conducted research into potential real-world continuous RL tasks.

REFERENCES

Shuai Bin Li. ShuaibinLi/RL_CARLA: Train auto_car in CARLA simulator with RL algorithms(SAC). URL https://github.com/ShuaibinLi/RL_CARLA.

CARLA Team. CARLA Simulator. URL https://carla.org/.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. *International Conference on Learning Representations*, 1 2021. doi: 10.48550/arxiv.2101.05982. URL https://arxiv.org/abs/2101.05982v2.

Florin Gogianu, Tudor Berariu, Mihaela Rosca, Claudia Clopath, Lucian Busoniu, and Razvan Pascanu. Spectral Normalisation for Deep Reinforcement Learning: an Optimisation Perspective. *International Conference on Machine Learning*, 5 2021. doi: 10.48550/arxiv.2105.05246. URL https://arxiv.org/abs/2105.05246v1.

Google. Regularization for Sparsity: L1 Regularization — Machine Learning — Google Developers. URL https://developers.google.com/machine-learning/crash-course/regularization-for-sparsity/l1-regularization.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *International Conference on Machine Learning*, 5:2976–2989, 1 2018. doi: 10.48550/arxiv.1801.01290. URL https://arxiv.org/abs/1801.01290v2.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout Q-Functions for Doubly Efficient Reinforcement Learning. *International Conference on Learning Representations*, 10 2021. doi: 10.48550/arxiv.2110.02034. URL https://arxiv.org/abs/2110.02034v2.

Jens Kober, † J Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, 32(11), 2013. doi: https://doi.org/10.1177/0278364913495721.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. *International Conference on Learning Representations*, 2 2018. doi: 10.48550/arxiv.1802.05957. URL https://arxiv.org/abs/1802.05957v1.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. *Conference on Robot Learning*, 2021. URL https://leggedrobotics.github.io/legged_gym/.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature 2016 529:7587*, 529(7587):484–489, 1 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 8 2022. doi: 10.48550/arxiv.2208.07860. URL https://arxiv.org/abs/2208.07860v1.

Adam Stooke and Pieter Abbeel. ACCELERATED METHODS FOR DEEP REINFORCEMENT LEARNING. 2018.

Victor Zhong, Caiming Xiong, and Richard Socher. SEQ2SQL: GENERATING STRUCTURED QUERIES FROM NATURAL LANGUAGE USING REINFORCEMENT LEARNING. 2017.

A Appendix

A.1 Appendix A: All Methods on Mujoco Environments

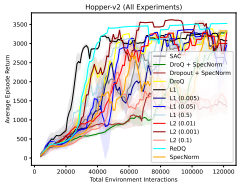
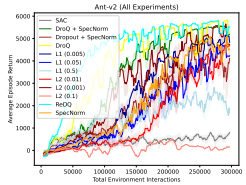
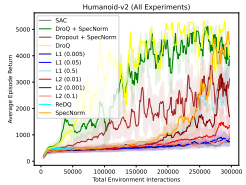
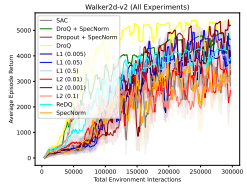
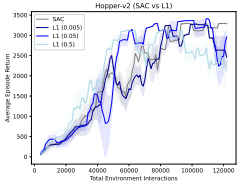
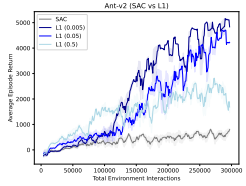
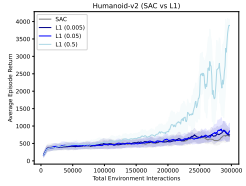
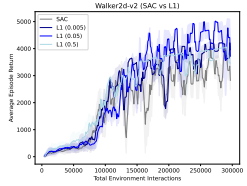
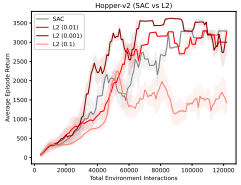
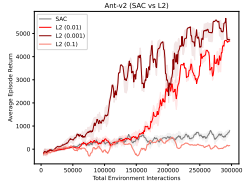
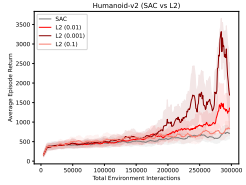
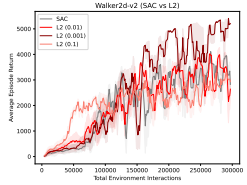
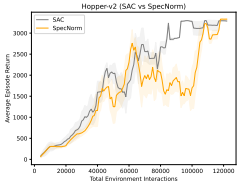
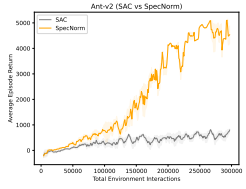
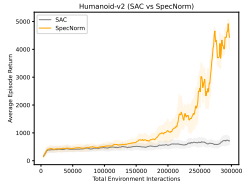
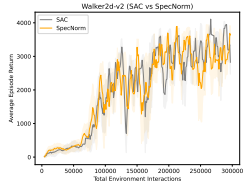
*	Hopper-v2	Ant-v2	Humanoid-v2	Walker2d-v2
(a) All				
(b) L1				
(c) L2				
(d) SpecNorm				

Table 1: All Experiments on Mujoco