Q-ENSEMBLE FOR OFFLINE RL: DON'T SCALE THE ENSEMBLE, SCALE THE BATCH SIZE

Anonymous authors
Paper under double-blind review

ABSTRACT

Training large neural networks is known to be time-consuming, with the learning duration taking days or even weeks. To address this problem, large-batch optimization was introduced. This approach demonstrated that scaling mini-batch sizes with appropriate learning rate adjustments can speed up the training process by orders of magnitude. While long training time was not typically a major issue for model-free deep offline RL algorithms, recently introduced Q-ensemble methods achieving state-of-the-art performance made this issue more relevant, notably extending the training duration. In this work, we demonstrate how this class of methods can benefit from large-batch optimization, which is commonly overlooked by the deep offline RL community. We show that scaling the mini-batch size and naively adjusting the learning rate allows for (1) a reduced size of the Q-ensemble, (2) stronger penalization of out-of-distribution actions, and (3) improved convergence time, effectively shortening training duration by 2.5x times on average.

1 INTRODUCTION

Offline Reinforcement Learning (ORL) provides a data-driven perspective on learning decision-making policies by using previously collected data without any additional online interaction during the training process Lange et al. (2012); Levine et al. (2020). Despite its recent development Fujimoto et al. (2019); Nair et al. (2020); An et al. (2021); Zhou et al. (2021); Kumar et al. (2020) and application Zhan et al. (2022); Apostolopoulos et al. (2021); Soares et al. (2021) progress, one of the current challenges in ORL remains algorithms extrapolation error, which is an inability to correctly estimate the values of unseen actions Fujimoto et al. (2019). Numerous algorithms were designed to address this issue. For example, Kostrikov et al. (2021) (IQL) avoids estimation for out-of-sample actions entirely. Similarly, Kumar et al. (2020) (CQL) penalizes out-of-distribution actions such that their values are lower-bounded. Other methods explicitly make the learned policy closer to the behavioral one Fujimoto & Gu (2021); Nair et al. (2020); Wang et al. (2020).

In contrast to prior studies, recent works An et al. (2021) demonstrated that simply increasing the number of value estimates in the Soft Actor-Critic (SAC) Haarnoja et al. (2018) algorithm is enough to advance state-of-the-art performance consistently across various datasets in the D4RL benchmark Fu et al. (2020). Furthermore, An et al. (2021) showed that the double-clip trick actually serves as an uncertainty-quantification mechanism providing the lower bound of the estimate, and simply increasing the number of critics can result in a sufficient penalization for out-of-distribution actions. Despite its state-of-the-art results, the performance gain for some datasets requires significant computation time or optimization of an additional term, leading to extended training duration (Figure 1).

In this paper, inspired by parallel works on reducing the training time of large models in other areas of deep learning You et al. (2019; 2017) (commonly referred to as large batch optimization), we study the overlooked use of large batches [1] in the deep ORL setting. We demonstrate that, instead of increasing the number of critics or introducing an additional optimization term in SAC-N algorithm, simple batch scaling and naive adjustment of the learning rate can (1) provide a sufficient penalty on out-of-distribution actions and (2) match state-of-the-art performance on the D4RL benchmark.

[1] Offline RL is often referred to as Batch RL, here, we extensively use the *batch* term to denote a mini-batch, not the dataset size.

Moreover, this large batch optimization approach significantly reduces the convergence time, making it possible to train models in under two hours using the worst-case dataset on a single-GPU setup. To the best of our knowledge, this is the first study that examines large batch optimization in the ORL setup.

2 How Long Does It Take for Q-Ensemble to Converge?

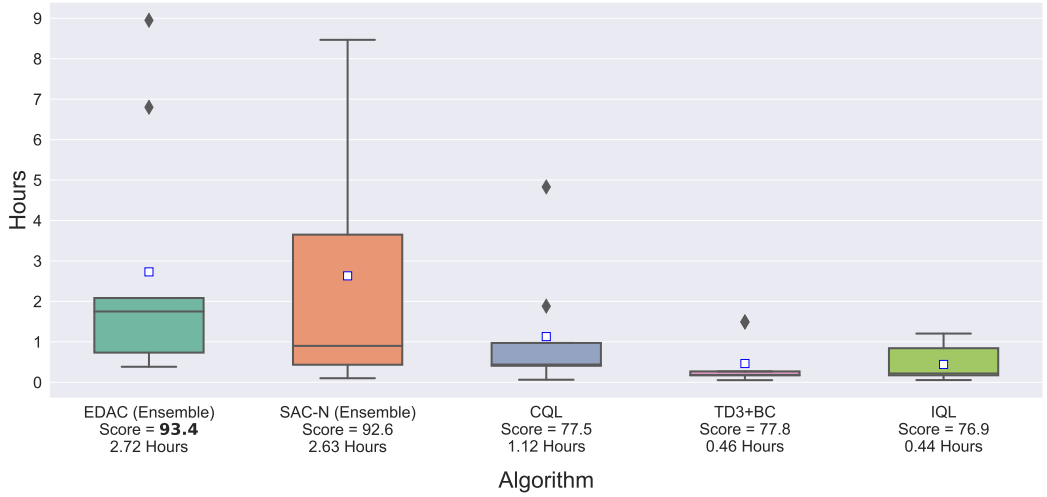


Figure 1: The convergence time of popular deep offline RL algorithms on 9 different D4RL locomotion datasets Fu et al. (2020). White boxes denote mean values (which are also illustrated on the x-axis). Note that the convergence time of Q-ensemble based methods is significantly higher.

To demonstrate the inflated training duration of Q-ensemble methods, we start with an analysis of their convergence time compared to other deep offline RL algorithms. Here, we show that while these methods consistently achieve state-of-the art performance across many datasets, the time it takes to obtain such results is significantly longer when compared to their closest competitors.

While it is not a common practice to report training times or convergence speed, some authors carefully analyzed these characteristics of their newly proposed algorithms Fujimoto & Gu (2021); Reid et al. (2022). There are two prevailing approaches for this analysis. First is to study the *total training time* for a fixed number of training steps or the speed per training step Fujimoto & Gu (2021); An et al. (2021). Second approach, such as the one taken by Reid et al. (2022), is to measure the *convergence time* using a relative wall-clock time to achieve the results reported in the performance tables.

Measuring the total training time for a fixed number of steps can be considered a straightforward approach. However, it does not take into account that some algorithms may actually require a smaller (or bigger) number of learning steps. Therefore, in this paper, we choose the second approach, which involves measuring the relative clock-time until convergence. Similar to Reid et al. (2022), we deem an algorithm to converge when its evaluation (across multiple seeds) achieves a normalized score within 2 points or higher of the one reported for the corresponding method in the performance table. In our analysis, we rely on the original implementations. We re-run the experiments to make sure that they are executed on the same hardware (Tesla A100) in order for the relative-clock time to be comparable.

The results are depicted in Figure 1. One can see that both IQL and TD3+BC are highly efficient compared to their competitors, and their convergence times do not exceed two hours even in the worst case scenarios. This efficiency comes from the fact that these methods do not take long for one training step Fujimoto & Gu (2021); Kostrikov et al. (2021). However, while being fast to train, they severely underperform when compared to the Q-ensemble based methods (SAC-N, EDAC).

Unfortunately, this improvement comes at a cost. For example, SAC-N median convergence time is under two hours (2-3x times longer than for IQL or TD3+BC), but some datasets may require up to

nine hours of training. This is due to its usage of a high number of critics, which requires more time for both forward and backward passes. Notably, EDAC was specifically designed to avoid using a large number of critics, reporting smaller computational costs per training step An et al. (2021) and lower memory consumption. However, as can be seen in Figure 1, its convergence time is still on par with the SAC-N algorithm. As we discussed earlier, some algorithms might require a higher number of training iterations to converge, which is exactly the case for the EDAC algorithm.

3 LARGE BATCH SOFT ACTOR-CRITIC

One common approach for reducing training time of deep learning models is to use large-batch optimization Hoffer et al. (2017); You et al. (2017; 2019). In this section, we investigate a similar line of reasoning for deep ORL and propose several adjustments to the SAC-N algorithm An et al. (2021) in order to leverage large mini-batch sizes:

1. Scale mini-batch Instead of using a significantly larger number of critics, as is done in SAC-N, we instead greatly increase the batch size from the typically used 256 state-action-reward tuples to 10k. Note that in the case of commonly employed D4RL datasets and actor-critic architectures, this increase does not require using more GPU devices and can be executed on the single-GPU setup. While higher batch sizes are also viable, we will demonstrate in Section 4 that this value is sufficient, and does not result in over-conservativeness.

2. Learning rate square root scaling In other research areas, it was often observed that a simple batch size increase is not enough to speed up convergence, and other additions are needed Hoffer et al. (2017). The learning rate adjustment is one such modification. Here, we use the Adam optimizer Kingma & Ba (2014), fixing the learning rate derived with a formula similar to Krizhevsky (2014) also known as "square root scaling":

$$learning\ rate = base\ learning\ rate * \sqrt{\frac{Batch\ Size}{Base\ Batch\ Size}}$$

where we take both $base\ learning\ rate$ and $base\ batch\ size$ to be equal to the values used in the SAC-N algorithm. Note that they are the same across all datasets. The specific values can be found in the Appendix A.7. Unlike Hoffer et al. (2017), we do not use a warm-up stage.

We refer to the described modifications as Large-Batch SAC (**LB-SAC**)[2]. Overall, the resulting approach is equivalent to the SAC-N algorithm, but with carefully adjusted values which are typically considered hyperparameters.

4 EXPERIMENTS

In this section, we present the empirical results of comparing LB-SAC with other Q-ensemble methods. We demonstrate that LB-SAC significantly improves convergence time while matching the final performance of other methods, and analyze what contributes to such performance.

4.1 EVALUATION ON THE D4RL MUJOCO GYM TASKS

We run our experiments on the commonly used MuJoCo locomotion subset of the D4RL benchmark Fu et al. (2020): Hopper, HalfCheetah, and Walker2D. Similar to the experiments conducted in Section 2, we use the original implementations for each studied algorithm.

LB-SAC Normalized Scores First, we report the final scores of the introduced approach. The results are illustrated in Table 1. One can see that the resulting scores match the original SAC-N algorithm. We highlight this result, as it is a common observation in large-batch optimization that naive learning rate adjustments typically lead to learning process degradation, making adaptive approaches or a warm-up stage necessary Hoffer et al. (2017); You et al. (2017; 2019). Evidently, this kind of treatment can be bypassed in the ORL setting for the SAC-N algorithm.

[2]One may rightfully point out that, while we work in a single-GPU setup, we rely on modern computational devices such as A100 with large memory capabilities. However, the memory consumption on D4RL datasets does not exceed 5GB of video memory in the worst-case scenarios (see A.4)

Table 1: Normalized average returns on D4RL Gym tasks, averaged over 4 random seeds. All the scores are taken from the original papers. LB-SAC attains a similar performance to both the SAC-N and EDAC algorithm while being considerably faster to converge as depicted in Figure 2.

Task Name	**TD3+BC**	**IQL**	**CQL**	**SAC-N**	**EDAC**	**LB-SAC**
halfcheetah-medium	48.3	47.4	58.5	67.5±1.2	65.9±0.6	67.0±0.9
halfcheetah-medium-expert	90.7	86.7	105.4	107.1±2.0	106.3±0.6	106.1±1.8
halfcheetah-medium-replay	44.6	44.2	95.0	63.9±0.8	61.3±1.9	64.0±0.8
hopper-medium	59.0	66.3	86.6	100.3±0.3	101.6±0.6	101.0±0.4
hopper-medium-expert	98.0	91.5	111.0	110.1±0.3	110.7±0.1	110.1±0.2
hopper-medium-replay	60.9	94.7	48.6	101.8±0.5	101.0±0.5	100.2±0.9
walker2d-medium	83.7	78.3	79.5	87.9±0.2	92.5±0.8	90.1±0.4
walker2d-medium-expert	110.1	109.6	109.1	116.7±0.4	114.7±0.9	116.4±0.5
walker2d-medium-replay	81.8	73.9	76.8	78.7±0.7	87.1±2.3	79.9±0.5
Average	75.2	76.9	85.6	92.6	**93.4**	**92.7**

LB-SAC Convergence Time Second, we study the convergence time in a similar fashion to Section 2. The results are depicted in Figure 2. One can see that the training times of LB-SAC are less than those of both EDAC and SAC-N, outperforming them in terms of average and the worst convergence times. Moreover, LB-SAC even outperforms CQL in the worst-case scenario, achieving commensurate average convergence time. This improvement comes from both (1) the usage of low number of critics, similar to EDAC (see Figure 3) and (2) improved ensemble convergence rates, which we demonstrate further in Section 4.2.

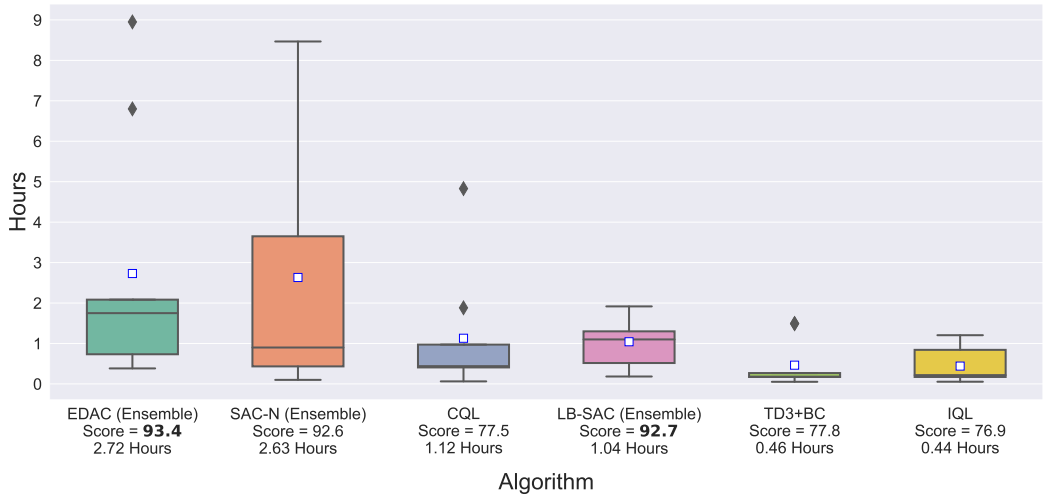


Figure 2: The convergence time of LB-SAC in comparison to other deep offline RL algorithms on 9 different D4RL locomotion datasets Fu et al. (2020). We consider an algorithm to converge similar to Reid et al. (2022), i.e., marking convergence at the point of achieving results similar to those reported in performance tables. White boxes denote mean values (which are also illustrated on the x-axis). LB-SAC shows significant improvement distribution-wise in comparison to both SAC-N and EDAC, notably performing better on the worst-case dataset even when compared to the ensemble-free CQL algorithm.

4.2 Bigger Batch Sizes Penalize OOD Actions

As argued in An et al. (2021), the major factor contributing to the success of Q-ensemble based methods in deep ORL setups is the penalization of actions based on prediction confidences. This is achieved by optimizing the lower bound of the value function. Both increasing the number of

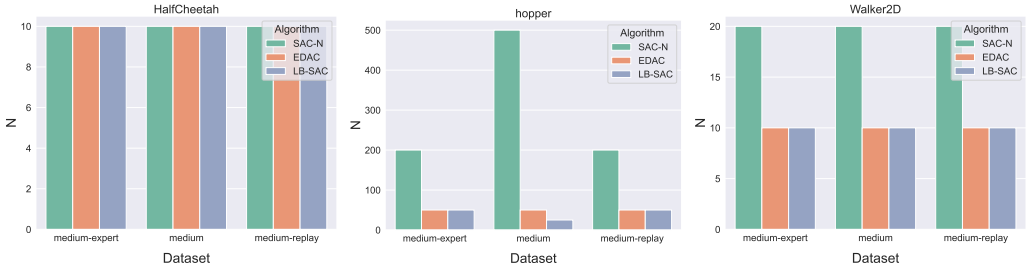
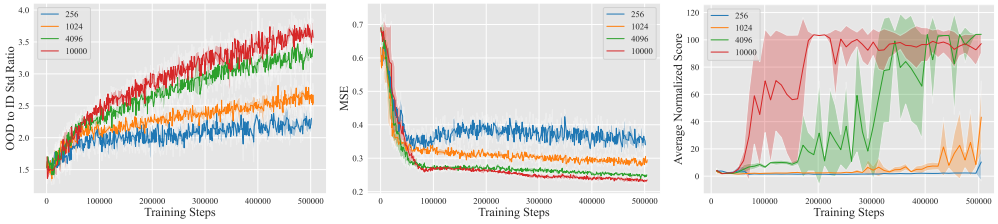


Figure 3: The number of Q-ensembles (N) used to achieve the performance reported in Table 1 and the convergence times in Figure 2. LB-SAC makes it possible to use a smaller number of members without the need to introduce an additional optimization term. Note that, for both SAC-N and EDAC, we used the number of critics described in the appendix of the original paper An et al. (2021).

critics and diversifying their outputs can improve prediction confidence, which in turn leads to enhanced performance An et al. (2021). On the other hand, unlike SAC-N, the proposed LB-SAC method does not rely on a large amount of critics (see Figure 3), or an additional optimization term for diversification as required by EDAC, while matching their performance with an improved convergence rate.



(a) Standard Deviation Relation (b) Distance to In-Dataset Actions (c) Normalized Score

Figure 4: Effects of increasing the batch size. **(a)** Depicts the relation of standard deviation for random actions to dataset actions **(b)** Plots the distance to the dataset actions from the learned policy **(c)** Shows averaged normalized score. The number of critics is fixed to 10 for all the batch sizes. Result averaged over 4 seeds. For the brevity of presentation, the results are reported for the hopper-medium-v2 dataset, however, a similar behavior occurs on all other datasets as well.

To explain the empirical success of LB-SAC, we vary the batch size and analyze the learning dynamics in a similar fashion to An et al. (2021) both in terms of out-of-distribution penalization and the resulting conservativeness, which is currently known to be the main observable property of successful ORL algorithms Kumar et al. (2020); Rezaeifar et al. (2022). To do so, we keep track of two values. First, we calculate the standard deviation of the q-values for both random and behavioral policies and record its relation throughout the learning process. The former is often utilized as the one producing out-of-distribution actions Kumar et al. (2020); An et al. (2021). Second, we estimate the distance between the actions chosen by the learned policy to the in-dataset ones for tracking how conservative the resulting policies are.

The learning curves are depicted in Figure 4. We observe that increasing the batch size results in faster growth of the standard deviation relation between out-of-distribution and in-dataset actions, which can produce stronger penalization on out-of-distribution actions earlier in the training process. This, in turn, leads to an elevated level of conservativeness, as demonstrated in Figure 4b. One possible explanation for why earlier penalization might be helpful is prevention of the iterative error exploitation (which is prominent in iterative algorithms and extensively discussed in Brandfonbrener et al. (2021)), as it allows to soften the overestimation bias early in the training process making its propagation less severe.

REFERENCES

Igor Adamski, Robert Adamski, Tomasz Grel, Adam Jędrych, Kamil Kaczmarek, and Henryk Michalewski. Distributed deep reinforcement learning: Learn how to play atari games in 21 minutes. In *International conference on high performance computing*, pp. 370–388. Springer, 2018.

Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6650–6656. IEEE, 2021.

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in neural information processing systems*, 34:7436–7447, 2021.

Pavlos Athanasios Apostolopoulos, Zehui Wang, Hanson Wang, Chad Zhou, Kittipat Virochsiri, Norm Zhou, and Igor L Markov. Personalization for web-based services using offline reinforcement learning. *arXiv preprint arXiv:2102.05612*, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

David Brandfonbrener, Will Whitney, Rajesh Ranganath, and Joan Bruna. Offline rl without off-policy evaluation. *Advances in Neural Information Processing Systems*, 34:4933–4946, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2021.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Vladislav Kurenkov and Sergey Kolesnikov. Showing your offline reinforcement learning work: Online evaluation budget matters. In *International Conference on Machine Learning*, pp. 11729–11752. PMLR, 2022.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pp. 45–73. Springer, 2012.

Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6131–6141. PMLR, 2021.

Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pp. 1702–1712. PMLR, 2022.

Nir Levine, Tom Zahavy, Daniel J Mankowitz, Aviv Tamar, and Shie Mannor. Shallow updates for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

Shideh Rezaeifar, Robert Dadashi, Nino Vieillard, Léonard Hussenot, Olivier Bachem, Olivier Pietquin, and Matthieu Geist. Offline reinforcement learning as anti-exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8106–8114, 2022.

Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *arXiv preprint arXiv:2111.03788*, 2021.

Hassam Sheikh, Kizza Frisbee, and Mariano Phielipp. DNS: Determinantal point process based neural network sampler for ensemble reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 19731–19746. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/sheikh22a.html`.

Douglas W Soares, Acordo Certo, Telma Lima, and Deep Learning Brazil. Pulserl: Enabling offline reinforcement learning for digital marketing systems via conservative q-learning. *Advances in Neural Information Processing Systems, 2nd Workshop on Offline Reinforcement Learning*, 2021.

Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations*, 2019.

Xianyuan Zhan, Haoran Xu, Yue Zhang, Xiangyu Zhu, Honglei Yin, and Yu Zheng. Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4680–4688, 2022.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, pp. 1719–1735. PMLR, 2021.

A Appendix

A.1 Related Work

Model-Free Offline RL A large portion of recently proposed deep offline RL algorithms focuses on addressing the extrapolation issue, trying to impose a certain degree of conservatism limiting the deviation of the final policy from the behavioral one. Researchers approached this problem from multiple angles. For example, Kumar et al. (2020) proposed to directly penalize out-of-distribution actions, while Kostrikov et al. (2021) avoids estimating values for out-of-sample actions completely. Others Fujimoto & Gu (2021); Nair et al. (2020); Wang et al. (2020) put explicit constraints to stay closer to the behavioral policy either by directly making models closer to the behavioral one Fujimoto & Gu (2021), or by re-weighting behavioral policy actions with the estimated advantages. In addition, there are works that construct a latent policy action space and optimize models within it Zhou et al. (2021); Allshire et al. (2021). In our work, we explore a different class of methods based on a model-free uncertainty quantification An et al. (2021), and demonstrate that it can be achieved with a sufficiently large mini-batch sizes and a handful number of value estimates.

Q-Ensembles in RL There has been a body of work on leveraging ensembles in deep reinforcement learning Sheikh et al. (2022); An et al. (2021); Lee et al. (2021; 2022); Osband et al. (2018). For offline RL, An et al. (2021) demonstrated that using the soft-actor critic with a large amount of value estimates leads to state-of-the art results on D4RL benchmark. As the number of critics grows, so do the computational time, in order to remedy this issue, An et al. (2021) further proposed to diversify the ensemble by disaligning the critics. In our work, we largely build upon these findings. However, we demonstrate that it is possible to alleviate a large number of critics by simply using large mini-batch sizes, which stabilizes the training process and speeds up convergence both in terms of training cycles and wall time.

Large Batch Optimization Unlike deep offline RL, the effect of larger mini-batch sizes was extensively studied in other areas of deep learning. This is typically referred as large batch optimization Hoffer et al. (2017). For example, You et al. (2017) proposed to use layer-wise adaptive learning rates to train big vision models in minutes. In the Natural Language Processing field, You et al. (2019) proposed another adaptive learning rates mechanism to train attention-based models such as BERT, making it possible to train big language models within hours instead of days. In our work, we investigate a similar line of reasoning, and demonstrate that simply increasing the mini-batch size and naively adjusting the learning rate is sufficient for obtaining a faster convergence speed while matching state-of-the-art performance on the D4RL benchmark.

Large Batch in RL While large batches are not commonly employed in online RL, their benefits were already extensively demonstrated and analyzed. For example, Berner et al. (2019) describes a large-scale setting and uses over a million timesteps per update to train a Dota-agent. On the other hand, in simple Atari environments, batch sizes of thousands were also found to be effective for online RL algorithms Levine et al. (2017); Horgan et al. (2018); Adamski et al. (2018); Stooke & Abbeel (2018). Notably, McCandlish et al. (2018) proposed a statistic called "gradient noise scaling" to predict the largest useful batch size for both supervised and online reinforcement learning problems. As for offline RL, to the best of our knowledge, we are the first to explore the effects of large mini-batch sizes and demonstrate how it can benefit the learning process of Q-ensemble methods.

A.2 Limitations and Future Work

Multiple Evaluations In our study we extensively use the notion of convergence time by evaluating multiple checkpoints during the training process. This contradicts the penultimate limitation of

pure offline RL, where one should be allowed to evaluate exactly one policy. However, in this work, we were specifically interested in studying the convergence properties, therefore requiring multiple evaluations for such an analysis. Still, the practice of multiple evaluations is common in the community Kurenkov & Kolesnikov (2022); Seno & Imai (2021) since the proper approach to compare offline RL methods is still an open question.

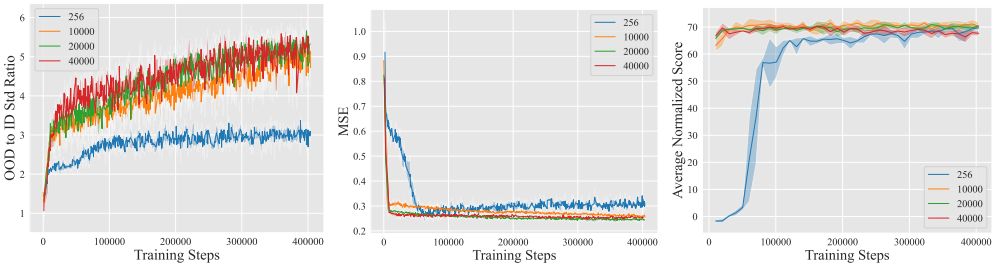
Overfitting It was noted in Seno & Imai (2021) that the best checkpoint across the training process significantly outperforms the final performance typically reported in deep offline RL papers. This is either attributed to the noisy learning process or some form of overfitting, as it can be often observed that the performance starts to deteriorate with more learning steps Seno & Imai (2021). We observed a similar behavior for large-batch optimization on some datasets (see Figure 5 for 40k batch size). We believe this is an interesting direction for future work (including large-batch settings), since the methodological apparatus for studying the properties of overfitting is yet to be developed for the offline RL setting.

LBO for Other Offline RL Algorithms Although the sole focus of our work was on studying large-batch optimization for Q-ensemble methods to demonstrate the benefits it can bring (e.g. a reduced number of critics or improved convergence time), an intriguing line of investigation would be the application of big batch sizes to other deep offline RL algorithms which could potentially also lead to improved convergence.

Layeradaptive Optimizers for Offline RL While we conducted a preliminary set of experiments on layer-adaptive optimizers You et al. (2017; 2019), more investigation is certainly needed in this direction. Since we run our experiments on D4RL benchmark and locomotion environments, we believe these optimizers may find their use in the context of different tasks typically employing other types of architecture (e.g. convolutional neural networks, recurrent networks, and transformers).

A.3 Ablation

A.3.1 How Large Should the Batch Be?



(a) Standard Deviation Relation (b) Distance to In-Dataset Actions (c) Normalized Score

Figure 5: Scaling batch size even further leads to diminishing returns. **(a)** Depicts the relation of standard deviation for random actions to dataset actions **(b)** Plots the distance between dataset actions and the learned policy **(c)** Shows the averaged normalized score. The number of critics is fixed for all batch sizes. Results were averaged over 4 seeds. For presentation brevity, the results are reported for the halfcheetah-medium-v2 dataset. However, similar behavior occurs on all other datasets as well.

Observing that the increased mini-batch size improves the convergence, a natural questions to ask is whether should we scale it even bigger. To answer this, we run LB-SAC with batch-sizes up to 40k.

We analyze the learning dynamics in a similar way to Section 4.2. The learning curves can be found in the Figure 5. Overall, we see that further increase leads to diminishing returns on both penalization and the resulting score. For example, Figure 5a shows that increasing the batch further leads to an improved penalization of the out-of-distribution actions. However, this improvement becomes less pronounced as we go from 20k to 40k. It may be interesting to note that the normalized scores for the typically utilized batch size of 256 start to rise after a certain level of penalization, and conservatism (MSE is around 0.3) is achieved. Evidently, the utilization of larger batch sizes helps achieve this level significantly earlier in the learning process.

A.3.2 Is It Just the Learning Rate?

To illustrate that the attained convergence rates benefit from an increased batch size and not merely from an elevated learning rate, we conduct an ablation, where we leave size of the batch equivalent to the one used in the SAC-N algorithm ($B = 256$), but inflate the learning rate. The results are depicted in Figure 6.

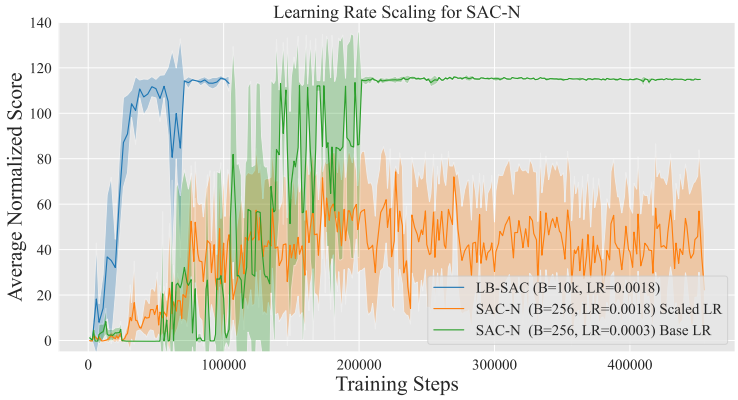
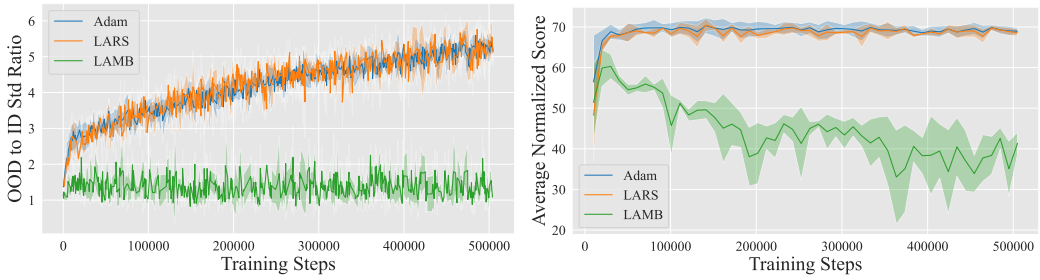


Figure 6: The average normalized score. The improvement comes from both large batch size and adjusted learning rate. Fixing batch size to the values typically used for SAC-N and scaling the learning rate does not help. The results are averaged over 4 seeds. For presentation brevity, the results are reported for the walker2d-medium-expert-v2 dataset.

Unsurprisingly, scaling learning rate without doing the same for batch size does not result in effective policies and stagnates at some point during the training process. This can be considered expected, as the hyperparameters for the base SAC algorithm were extensively tuned by an extensive number of works on off-policy RL and seem to adequately transfer to the offline setting, as described in An et al. (2021).

A.3.3 Do Layer-Adaptive Optimizers Help?



(a) Standard Deviation Relation

(b) Normalized Score

Figure 7: Using layer-adaptive optimizers leads either to a similar performance or degrades the learning process. **(a)** Depicts the relation of standard deviation for random actions to dataset actions **(b)** Shows the averaged normalized score. The number of critics is fixed for all optimizers. The results are averaged over 4 seeds. For presentation brevity, the results are reported for the halfcheetah-medium-v2 dataset.

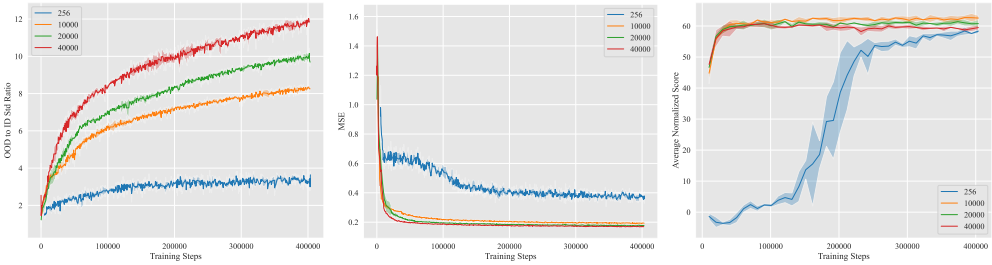
While we settled on a naive approach for large-batch optimization in the setting of deep ORL, one may wonder whether using more established and sophisticated optimizers may work as well. Here, we take a look at commonly employed LARS You et al. (2017) and LAMB You et al. (2019) optimizers reporting learning dynamics similar to the previous sections. The resulting curves can be found in Figure 7.

We see that the LAMB optimizer could not succeed in the training process diverging with more learning steps. On the other hand, LARS behaves very similar to the proposed adjustments. While by no means we suggest that adaptive learning rates are not useful for offline RL, the straight adoption of established large-batch optimizers for typically used neural network architecture (linear layers with activations) in the context of locomotion tasks does not bring much benefits. Seemingly, these methods may require more hyperparameter tuning, different learning rate scaling rules or a warm-up schedule as opposed to the simple "square root scaling".

A.3.4 On Layer Normalization

While scaling batch size significantly improves convergence speed and penalization of OOD actions, there were some difficult tasks where such naive approach could not help as much. As we showed in 4.2, scaling has its limits and can lead to diminishing returns, where at some point (around 10k batch size) the std ratio (as well as penalization) will stop growing. From this point, increasing batches further does not always provide much benefit, since without a sufficient level of penalization for a given task even with a faster convergence rate the agent may not converge or the training can be unstable. With a default MLP critic architecture, the only option left is to make the ensemble bigger or use diversity loss from EDAC. In our preliminary experiments, we found that increasing the ensemble size could remedy this issue but led to longer training times.

After more experimentation, we found that the use of Layer Normalization Ba et al. (2016) completely reduces diminishing returns on all batch sizes we considered (Figure 8). One can see in Figure 8 that the scaling tendency is preserved in opposition to the results we observed without layer normalization (Figure 5).



(a) Standard Deviation Relation (b) Distance to In-Dataset Actions (c) Normalized Score

Figure 8: Layer normalization reduces the effects of diminishing returns. **(a)** Depicts the relation of standard deviation for random actions to dataset actions **(b)** Plots the distance to the dataset actions from the learned policy **(c)** Shows the averaged normalized score. The number of critics is fixed for all batch sizes. Results were averaged over 4 seeds. halfcheetah-medium-v2 is the dataset used.

The most illustrative example is the hopper-medium-v2 dataset, which can be considered the hardest task among all others. To solve it, SAC-N uses 500 critics while EDAC uses 50 An et al. (2021). In our experiments, we were unable to get competitive results with bigger batches and 50 critics, and increasing ensemble size was not an option due to efficiency reasons. However, with additional layer normalization for critics, we were able to get strong results using only 25 critics due to increased penalization. Note that on more simple tasks (Figure 8), such a boost can lead to overconservative policies and lower scores as a result (almost 10 points worse than in 4.2).

A.4 LB-SAC Memory Consumption

Memory consumption is an obvious caveat of using large batch sizes. Here, we report the worst-case memory requirements for all of the Q-ensemble algorithms. Table 2 shows that utilization of large batches can indeed lead to a considerable increase in memory usage. However, this still makes it possible to employ LB-SAC even using single-GPU setups on devices with moderate computa-

Table 2: Worst-case number of critics and memory consumption for D4RL locomotion datasets.

	Critics (Quantity)	**GPU Mem.** (GB)
SAC-N	500	5.1
EDAC	50	1.8
LB-SAC	50	5.4

tional power, requiring around 5GB of memory in the worst-case scenarios.

A.5 Convergence Time

Table 3: Convergence times in minutes for each dataset.

Task Name	TD3+BC	IQL	CQL	SAC-N	EDAC	LB-SAC
halfcheetah-medium	3.1	6.5	20.6	26.0	84.0	11.0
halfcheetah-medium-expert	89.5	13.0	289.7	336.0	537.0	43.0
halfcheetah-medium-replay	89.5	3.2	24.4	17.0	23.0	28.0
hopper-medium	10.7	50.5	3.7	508.0	105.0	115.0
hopper-medium-expert	10.2	51.0	58.3	219.0	408.0	114.0
hopper-medium-replay	11.1	72.2	112.9	54.0	44.0	66.0
walker2d-medium	16.1	11.4	24.4	39.0	115.0	77.0
walker2d-medium-expert	9.3	19.5	48.9	216.0	125.0	78.0
walker2d-medium-replay	10.2	10.3	26.3	6.0	33.0	31.0
Average	27.7	**26.4**	67.6	157.8	163.7	**62.5**

A.6 Difference in Approaches

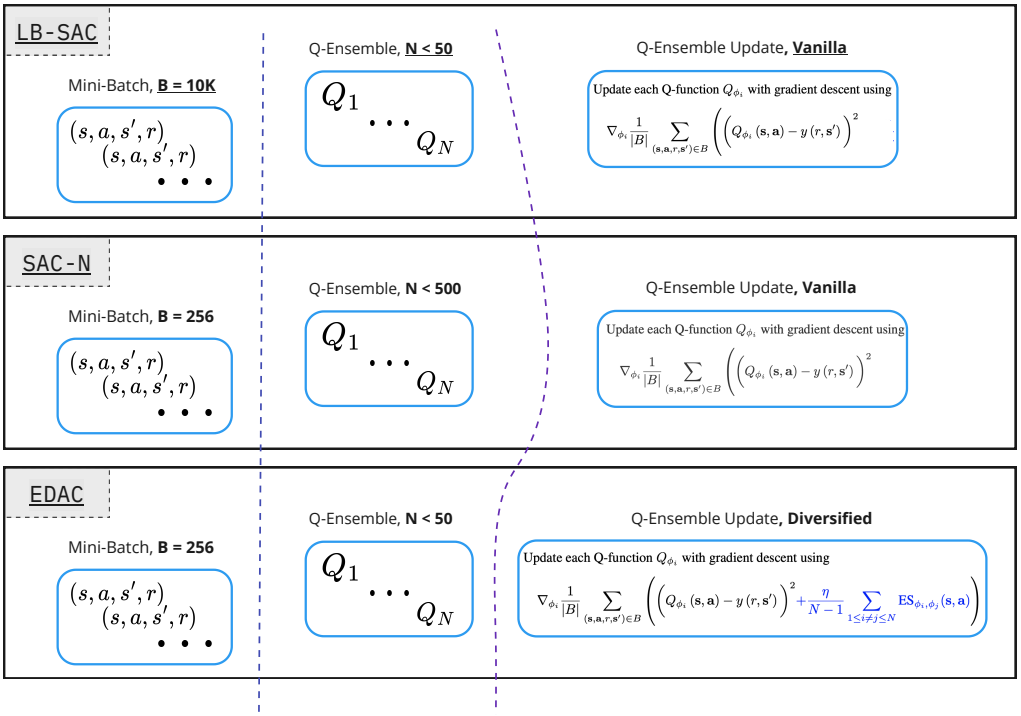


Figure 9: The difference between recently introduced SAC-N, EDAC, and the proposed LB-SAC. The introduced approach does not require an auxiliary optimization term while making it possible to effectively reduce the number of critics in the Q-ensemble by switching to the large-batch optimization setting.

A.7 HYPERPARAMETERS

Table 4: SAC-N, LB-SAC, EDAC shared hyperparameters

Parameter	**Value**
optimizer	AdamW (Loshchilov & Hutter, 2017)
tau (τ)	5e-3
hidden dim (all networks)	256
hidden layers (all networks)	3
target entropy	-action_dim
gamma (γ)	0.99
nonlinearity	ReLU

Table 5: Algorithm specific hyperparameters. For **LB-SAC** algorithm on all environments we used **SAC-N** batch size and learning rate as base values for learning rate scaling (precise formula described in 3).

Algorithm	**Batch size**	**Learning rate**
SAC-N	256	3e-4
EDAC	256	3e-4
LB-SAC	10000	1.8e-3

Table 6: Environment specific hyperparameters

Task Name	**SAC-N** (N)	**LB-SAC** (N, LayerNorm)	**EDAC** (N, η)
halfcheetah-medium	10	10, False	10, 1.0
halfcheetah-medium-expert	10	10, False	10, 5.0
halfcheetah-medium-replay	10	10, False	10, 1.0
hopper-medium	500	25, True	50, 1.0
hopper-medium-expert	200	50, False	50, 1.0
hopper-medium-replay	200	50, False	50, 1.0
walker2d-medium	20	10, False	10, 1.0
walker2d-medium-expert	20	10, False	10, 5.0
walker2d-medium-replay	20	10, False	10, 1.0