Why So Pessimistic? Estimating Uncertainties for Offline RL through Ensembles, and Why Their Independence Matters

Seyed **Kamyar** Seyed **Ghasemipour**
Robotics@Google
kamyar@google.com

Shixiang Shane Gu
Google Research, Brain Team
shanegu@google.com

Ofir Nachum
Google Research, Brain Team
ofirnachum@google.com

Abstract

Motivated by the success of ensembles for uncertainty estimation in supervised learning, we take a renewed look at how ensembles of Q-functions can be leveraged as the primary source of pessimism for offline reinforcement learning (RL). We begin by identifying a critical flaw in a popular algorithmic choice used by many ensemble-based RL algorithms, namely the use of shared pessimistic target values when computing each ensemble member's Bellman error. Through theoretical analyses and construction of examples in toy MDPs, we demonstrate that shared pessimistic targets can paradoxically lead to value estimates that are effectively *optimistic*. Given this result, we propose MSG, a practical offline RL algorithm that trains an ensemble of Q-functions with independently computed targets based on completely separate networks, and optimizes a policy with respect to the lower confidence bound of predicted action values. Our experiments on the popular D4RL and RL Unplugged offline RL benchmarks demonstrate that on challenging domains such as antmazes, MSG with deep ensembles surpasses highly well-tuned state-of-the-art methods by a wide margin. Additionally, through ablations on benchmarks domains, we verify the critical significance of using independently trained Q-functions, and study the role of ensemble size. Finally, as using separate networks per ensemble member can become computationally costly with larger neural network architectures, we investigate whether efficient ensemble approximations developed for supervised learning can be similarly effective, and demonstrate that they do not match the performance and robustness of MSG with separate networks, highlighting the need for new efforts into efficient uncertainty estimation directed at RL.

1 Introduction

Offline reinforcement learning (RL), also referred to as batch RL [1], is a problem setting in which one is provided a dataset of interactions with an environment in the form of a Markov decision process (MDP), and the goal is to learn an effective policy exclusively from this fixed dataset. Offline RL holds the promise of data-efficiency through data reuse, and improved safety due to minimizing the need for policy rollouts. As a result, offline RL has been the subject of significant renewed interest in the machine learning literature [2].

One common approach to offline RL in the model-free setting is to use approximate dynamic programming (ADP) to learn a Q-value function via iterative regression to backed-up *target values*. The predominant algorithmic philosophy with most success in ADP-based offline RL is to encourage

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

obtained policies to remain close to the support set of the available offline data. A large variety of methods have been developed for enforcing such constraints, examples of which include regularizing policies with behavior cloning objectives [3, 4], performing updates only on actions observed inside [5, 6, 7, 8] or close to [9] the offline dataset, and regularizing value functions to underestimate the value of actions not seen in the dataset [10, 11, 12].

The need for such regularizers arises from inevitable inaccuracies in value estimation when function approximation, bootstrapping, and off-policy learning – i.e. The Deadly Triad [13] – are involved. In offline RL in particular, such inaccuracies cannot be resolved through additional interactions with the MDP. Thus, remaining close to the offline dataset limits opportunities for catastrophic inaccuracies to arise. However, recent works have argued that the aforementioned constraints can be overly pessimistic, and instead opt for approaches that take into consideration the *uncertainty* about the value function [14, 15, 16], thus re-focusing the offline RL problem to that of deriving accurate lower confidence bounds (LCB) of Q-values.

In the empirical supervised learning literature, *deep network ensembles* (definition in Appendix L) and their more efficient variants have been shown to be the most effective approaches for uncertainty estimation, towards learning calibrated estimates and confidence bounds with modern neural network function approximators [17]. Motivated by this, in our work we take a renewed look into Q-ensembles, and study how to leverage them as the primary source of pessimism for offline RL.

In deep RL, a very popular algorithmic choice is to use an ensemble of Q-functions to obtain pessimistic value estimates and combat overestimation bias [18]. Specifically, in the policy evaluation procedure, all Q-networks are updated towards a *shared pessimistic temporal difference target*. Similarly in offline RL, in addition to the main offline RL objective that they propose, several existing methods use such Q-ensembles [10, 3, 19, 20, 21, 22, 23, 8]. We begin by mathematically characterizing a critical flaw in the aforementioned ensembling procedure. Specifically, we demonstrate that using shared pessimistic targets can paradoxically lead to Q-estimates which are in fact *optimistic*! We verify our finding by constructing pedagogical toy MDPs. These results demonstrate that the formulation of using shared pessimistic targets is fundamentally ill-formed.

To resolve this problem, we propose *Model Standard-deviation Gradients* (MSG), an ensemble-based offline RL algorithm. In MSG, each Q-network is trained *independently*, *without sharing targets*. Crucially, ensembles trained with independent target values will always provide pessimistic value estimates. The pessimistic lower-confidence bound (LCB) value estimate – computed as the mean minus standard deviation of the Q-value ensemble – is then used to update the policy being trained. Evaluating MSG on the established D4RL [24] and RL Unplugged [25] benchmarks for offline RL, we demonstrate that MSG matches, and in the more challenging domains such as antmazes, significantly exceeds the prior state-of-the-art. Additionally, through a series of ablation experiments on benchmark domains, we verify the significance of our theoretical findings, study the role of ensemble size, and highlight the settings in which ensembles provide the most benefit.

The use of ensembles will inevitably be a computational bottleneck when applying offline RL to domains requiring large neural network models. Hence, as a final analysis, we investigate whether the favorable performance of MSG can be obtained through the use of modern *efficient ensemble* approaches which have been successful in the supervised learning literature [26, 27, 28, 17]. We demonstrate that while efficient ensembles are competitive with the state-of-the-art on simpler offline RL benchmark domains, similar to many popular offline RL methods they fail on more challenging tasks, and cannot recover the performance and robustness of MSG using full ensembles with separate neural networks.

Our work highlights some of the unique and often overlooked challenges of ensemble-based uncertainty estimation in offline RL. Given the strong performance of MSG, we hope our work motivates increased focus into efficient and stable ensembling techniques directed at RL, and that it highlights intriguing research questions for the community of neural network uncertainty estimation researchers whom thus far have not employed sequential domains such as offline RL as a testbed for validating modern uncertainty estimation techniques.

2 Related Work

Uncertainty estimation is a core component of RL, since an agent only has a limited view into the mechanics of the environment through its available experience data. Traditionally, uncertainty

estimation has been key to developing proper *exploration* strategies such as upper confidence bound (UCB) and Thompson sampling [29], in which an agent is encouraged to seek out paths where its uncertainty is high. Offline RL presents an alternative paradigm, where the agent must act conservatively and is thus encouraged to seek out paths where its uncertainty is low [14]. In either case, proper and accurate estimation of uncertainties is paramount. To this end, much research has been produced with the aim of devising provably correct uncertainty estimates [30, 31, 32], or at least bounds on uncertainty that are good enough for acting exploratorily [33] or conservatively [34]. However, these approaches require exceedingly simple environment structure, typically either a finite discrete state and action space or linear spaces with linear dynamics and rewards.

While theoretical guarantees for uncertainty estimation are more limited in practical situations with deep neural network function approximators, a number of works have been able to achieve practical success, for example using deep network analogues for count-based uncertainty [35], Bayesian uncertainty [36, 37], and bootstrapping [38, 39]. Many of these methods employ ensembles. In fact, in continuous control RL, it is common to use an ensemble of two value functions and use their minimum for computing a target value during Bellman error minimization [18]. A number of works in offline RL have extended this to propose backing up minimums or lower confidence bound estimates over larger ensembles [3, 10, 19, 20, 22, 23, 21]. In our work, we continue to find that ensembles are extremely useful for acting conservatively, but the manner in which ensembles are used is critical. Specifically our proposed MSG algorithm advocates for using independently learned ensembles, without sharing of target values, and this important design decision is supported by empirical evidence.

The widespread success of ensembles for uncertainty estimation in RL echoes similar findings in supervised deep learning. While there exist proposals for more technical approaches to uncertainty estimation [40, 41, 42], ensembles have repeatedly been found to perform best empirically [26, 43]. Much of the active literature on ensembles in supervised learning is concerned with computational efficiency, with various proposals for reducing the compute or memory footprint of training and inference on large ensembles [28, 44, 27]. While these approaches have been able to achieve impressive results in supervised learning, our empirical results suggest that their performance suffers significantly in challenging offline RL settings compared to deep ensembles.

3 Pessimistic Q-Ensembles: Independent or Shared Targets?

In this section we identify a critical flaw in how ensembles are commonly employed – in offline as well as online RL – for obtaining pessimistic value estimates [10, 3, 19, 20, 21, 22, 23, 8, 21], which can paradoxically lead to an optimism bonus! We begin by mathematically characterizing this problem and presenting a simple fix. Subsequently, we leverage our results to construct pedagogical toy MDPs demonstrating the practical importance of the identified problem and solution.

3.1 Mathematical Characterization

We assume access to a dataset D composed of (s, a, r, s') transition tuples from a Markov Decision Process (MDP) determined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, corresponding to state space, action space, reward function, transitions dynamics, and discount, respectively. As is standard in RL, we do not assume any knowledge of $\mathcal{R}, \mathcal{P}$, other than that implicitly provided by the dataset D. In this section, for clarity of exposition, we assume that the policies we consider are *deterministic*, and that our MDPs do not have terminal states.

1. Initialize θ^i for all $i \in Z$.
2. For $t = 1, 2, \ldots$:
   - For each $(s, a, r, s') \in D$ and $i \in Z$ compute target values $y^i(r, s', \pi)$.
   - For each $i \in Z$, update θ^i to optimize the regression objective
   $$\frac{1}{|D|} \sum_{(s,a,r,s') \in D} (Q_{\theta^i}(s, a) - y^i(r, s', \pi))^2$$
3. Return a pessimistic Q-value function $Q_{\text{pessimistic}}$ based on the trained ensemble.

We consider Q-value ensemble members given by a parameterization Q_{θ^i}, where i indexes into some set Z, which is finite in practice but may be infinite or uncountable in theory. We assume Z has an associated probability space allowing us to make expectation $\mathbb{E}$ or variance $\mathbb{V}$

computations over the ensemble members. Given a fixed policy π, a general dynamic programming based procedure for obtaining pessimistic value estimates is outlined by the iterative regression described in the box above.

A key algorithmic choice in this recipe is where pessimism should be introduced. This can be done by either (a) pessimistically aggregating Q-values after training, i.e. inside Step 3, or (b) *also incorporating pessimism during Step 2,* by using a shared pessimistic target value y. Through our review of the offline RL (as well as online RL) literature, we have observed that the most common approach is the latter, where the targets are *pessimistic, shared, and identical* across ensemble members [10, 3, 19, 20, 21, 22, 23, 8]. Specifically, they are computed as, $y^i(r, s', \pi) = \mathrm{PO}(\{r + \gamma Q_{\theta^i}(s', \pi(s')), \forall i \in Z\})$ with PO being a desired pessimism operator aggregating the TD target values of the ensemble members (e.g. "mean minus standard deviation", or "minimum").

In this section, our goal is to compare these two alternative approaches. For our analysis, we will use "mean minus standard deviation" (a lower confidence bound (LCB)) as our pessimism operator, and use the notation Q_{LCB} in place of $Q_{\mathrm{pessimistic}}$ (defined in the box above). Under the LCB pessimism operator we will have:

Independent Targets (Method 1): $y^i(r, s', \pi) = r + \gamma \cdot Q_{\theta^i}(s', \pi(s'))$

Shared Targets (Method 2): $y^i(r, s', \pi) = r + \gamma \cdot \left(\mathbb{E}_{\text{ens}}\left[Q_{\theta^i}(s', \pi(s'))\right] - \sqrt{\mathbb{V}_{\text{ens}}\left[Q_{\theta^i}(s', \pi(s'))\right]}\right)$

For both we have: $Q_{\mathrm{LCB}}(s, a) = \mathbb{E}_{\text{ens}}\left[Q_{\theta^i}(s, a)\right] - \sqrt{\mathbb{V}_{\text{ens}}\left[Q_{\theta^i}(s, a)\right]}$

To characterize the form of Q_{LCB} when using complex neural networks, we refer to the work on infinite-width neural networks, namely the Neural Tangent Kernel (NTK) [45]. We consider Q-value ensemble members, Q_{θ^i}, which all share the same infinite-width neural network architecture (and thus the same NTK parameterization). As noted in the algorithm box above, and as is the case in deep ensembles [43], the only difference amongst ensemble members Q_{θ^i} is in their initial weights θ^i sampled from the neural network's initial weight distribution.

Before presenting our results, we establish some notation relevant to the infinite-width and NTK regime. Let $\mathcal{X}, R, \mathcal{X}'$ denote data matrices containing (s, a), r, and $(s', \pi(s'))$ appearing in the offline dataset D; i.e., the k-th transition (s, a, r, s') in D is represented by the k-th rows in $\mathcal{X}, R, \mathcal{X}'$. Let A, B denote two data matrices, where similar to $\mathcal{X}, \mathcal{X}'$, each row contains a state-action tuple $(s, a) \in \mathcal{S} \times \mathcal{A}$. The NTK, which governs the training dynamics of the infinitely-wide neural network, is then given by the outer product of gradients of the neural network *at initialization*: $\hat{\Theta}_i^{(0)}(A, B) := \nabla_\theta Q_{\theta^i}(A) \cdot \nabla_\theta Q_{\theta^i}(B)^T|_{t=0}$, where we overload notation $Q_{\theta^i}(A)$ to represent the column vector containing Q-values. At infinite-width in the NTK regime, $\hat{\Theta}_i^{(0)}(A, B)$ converges to a deterministic kernel (i.e. does not depend on the random weight sample θ^i), and hence is the same for all ensemble members. Thus, hereafter we will remove the index i from the notation of the NTK kernel and simply write, $\hat{\Theta}^{(0)}(A, B)$. With our notation in place, we define, $C := \hat{\Theta}^{(0)}(\mathcal{X}', \mathcal{X}) \cdot \hat{\Theta}^{(0)}(\mathcal{X}, \mathcal{X})^{-1}$. Intuitively, C is a $|D| \times |D|$ matrix where the element at column q, row p, captures a notion of similarity between (s, a) in the qth row of $\mathcal{X}$, and $(s', \pi(s'))$ in the pth row of $\mathcal{X}'$.

We now have all the necessary machinery to characterize the form of Q_{LCB}:

Theorem 3.1. *For a given $(s, a) \in \mathcal{S} \times \mathcal{A}$, let $Q_{\theta^i}^{(0)}(s, a)$ denote $Q_{\theta^i}(s, a)|_{t=0}$ (value at initialization), with θ sampled from the initial weight distribution. After $t + 1$ iterations of pessimistic policy evaluation, the LCB value estimate for $(s', \pi(s')) \in \mathcal{X}'$ is given by,*

Independent Targets (Method 1): (1)

$$Q_{\mathrm{LCB}}^{(t+1)}(\mathcal{X}') = \mathcal{O}(\gamma^t \|C\|^t) + \underbrace{(1 + \ldots + \gamma^t C^t)}_{\textit{backup term}} CR - \sqrt{\mathbb{E}_{\text{ens}}\left[\left(\underbrace{(1 + \ldots + \gamma^t C^t)}_{\textit{backup term}}(Q_{\theta^i}^{(0)}(\mathcal{X}') - CQ_{\theta^i}^{(0)}(\mathcal{X}))\right)^2\right]}$$

Shared Targets (Method 2): (2)

$$Q_{\mathrm{LCB}}^{(t+1)}(\mathcal{X}') = \mathcal{O}(\gamma^t \|C\|^t) + \underbrace{(1 + \ldots + \gamma^t C^t)}_{\textit{backup term}} CR - \underbrace{(1 + \ldots + \gamma^t C^t)}_{\textit{backup term}} \sqrt{\mathbb{E}_{\text{ens}}\left[\left(Q_{\theta^i}^{(0)}(\mathcal{X}') - CQ_{\theta^i}^{(0)}(\mathcal{X})\right)^2\right]}$$

where the square and square-root operations are applied element-wise.[1] *Please refer to Appendix F for the proof.*

As can be seen, the equations for the pessimistic LCB value estimates in both settings are similar, only differing in the third term. The first term is negligible and tends towards zero as the number of iterations of policy evaluation increases. The second term shared by both variants corresponds to the expected result of the policy evaluation procedure *without any pessimism* (as before, we mean expectation under θ sampled from the initial weight distribution). Accordingly, the differing third term in each variant exactly corresponds to the "pessimism" or "penalty" induced by that variant.

Considering the available offline RL dataset D as a restricted MDP in itself, we see that the use of Independent Targets (Method 1) leads to a pessimism term that performs "backups" along the trajectories that the policy would experience in this restricted MDP (using the geometric term $1 + \cdots + \gamma^t C^t$) before computing a variance estimate. Meanwhile the use of Shared Targets (Method 2) does the reverse – it first computes a variance term and then performs the "backups".

While this difference may seem inconsequential, it becomes critical when one realizes that in Equation 2 for Shared Targets (Method 2), the pessimism term (third term) may become positive, i.e. a *negative penalty*, yielding an effectively *optimistic* LCB estimate. Critically, with Independent Targets (Method 1), this problem *cannot occur*.

3.2 Validating Theoretical Predictions

In this section we demonstrate that our analysis is not solely a theoretical result concerning the idiosyncracies of infinite-width neural networks, but that it is rather straightforward to construct combinations of an MDP, offline data, and a policy, that lead to the critical flaw of an optimistic LCB estimate.

1. Initialize empty $\mathcal{X}, R, \mathcal{X}'$
2. For N episodes:
   - sample $s \sim \mathcal{N}(0, \mathrm{I})$
   - For T steps:
     - sample $a \sim \mathcal{N}(0, \mathrm{I})$
     - sample $s' \sim \mathcal{N}(0, \mathrm{I})$
     - set $\pi(s') \leftarrow a$
     - Add (s, a) to $\mathcal{X}$
     - Add $r \sim \mathcal{N}(0, \mathrm{I})$ to R
     - Add $(s', \pi(s'))$ to $\mathcal{X}'$
     - Set $s \leftarrow s'$
3. Return the offline dataset $\mathcal{X}, R, \mathcal{X}'$

Let d_s, d_a denote the dimensionality of state and action vectors respectively. We consider an MDP whose initial state distribution is a spherical multivariate normal distribution $\mathcal{N}(0, \mathrm{I})$, and whose transition function is given by $\mathcal{P}(s'|s, a) = \mathcal{N}(0, \mathrm{I})$. Consider the procedure for generating our offline data matrices, described in the box to the right. This procedure returns data matrices $\mathcal{X}, R, \mathcal{X}'$ by generating N episodes of length T, using a behavior policy $a \sim \mathcal{N}(0, \mathrm{I})$. In this generation process, we set the policy we seek to pessimistically evaluate, π, to always apply the behavior policy's action in state s to the next state s'.

To construct our examples, we consider the setting where we use linear models to represent Q_{θ^i}, with the initial weight distribution being a spherical multivariate normal distribution, $\mathcal{N}(0, \mathrm{I})$. With linear models, the equations for Q_{LCB} takes an identical form to those in Theorem 3.1.

Given the described data generating process and our choice of linear function approximation, we can compute the pessimism term for the Shared Targets (Method 2) (i.e. the third term in Theorem 3.1, Equation 2). We implement this computation in a simple Python script, which we include in the supplementary material. We choose, $d_s = 30$, $d_a = 30$, $\gamma = 0.5$, $N = 5$, $T = 5$, and $t = 1000$ (t is the exponent in the geometric term above). We run this simulation 1000 times, each with a different random seed. After filtering simulation runs to ensure $\gamma\|C\| < 1$ (as discussed in an earlier footnote), we observe that 221 of the simulation runs result in an optimistic LCB bonus, meaning that in those experiments, the pessimism term was in fact positive for some $(s', \pi(s')) \in \mathcal{X}'$. We have made the python notebook implementing this experiment available in our supplementary material. For further intriguing investigations in pedagogical toy MDPs regarding the structure of uncertainties, we strongly encourage the interested reader to refer to Appendix G.

[1] Note that if $\gamma\|C\| \geq 1$, dynamic programming is liable to diverge in either setting. In our discussions, we avoid this degenerate case and assume $\gamma\|C\| < 1$.

4 Model Standard-deviation Gradients (MSG)

It is important to note that even if the pessimism term does not become positive for a particular combination of MDPs, offline datasets, and policies, the fact that it can occur highlights that the formulation of Shared Targets is fundamentally ill-formed. To resolve this problem we propose Model Standard-deviation Gradients (MSG), an offline RL algorithm which leverages ensembles to approximate the LCB using the approach of Independent Targets.

4.1 Policy Evaluation and Optimization in MSG

MSG follows an actor-critic setup. At the beginning of training, we create an ensemble of N Q-functions by taking N samples from the initial weight distribution. During training, in each iteration, we first perform policy evaluation by estimating the Q_{LCB} for the current policy, and subsequently optimize the policy through gradient ascent on Q_{LCB}.

Policy Evaluation As motivated by our analysis in Section 3, we train the ensemble Q-functions independently using the standard least-squares Q-evaluation loss,

$$\mathcal{L}(\theta^i) = \mathbb{E}_{(s,a,r,s')\sim D}\Big[\big(Q_{\theta^i}(s,a) - y^i(r,s',\pi)\big)^2\Big]; \quad y^i = r + \gamma \cdot \mathbb{E}_{a'\sim\pi(s')}\Big[Q_{\bar{\theta}^i}(s',a')\Big] \tag{3}$$

where $\theta^i, \bar{\theta}^i$ denote the parameters and target network parameters for the ith Q-function.

In each iteration, as is common practice, we do not update the Q-functions until convergence, and instead update the networks using a single gradient step. In practice, the expectation in $\mathcal{L}(\theta^i)$ is estimated by a minibatch, and the expectation in y^i is estimated with a single action sample from the policy. After every update to the Q-function parameters, their corresponding target parameters are updated to be an exponential moving average of the parameters in the standard fashion.

Policy Optimization As in standard deep actor-critic algorithms, policy evaluation steps (learning Q) are interleaved with policy optimization steps (learning π). In MSG, we optimize the policy through gradient ascent on Q_{LCB}. Specifically, our proposed policy optimization objective in MSG is,

$$\mathcal{L}(\pi) = \mathbb{E}_{s\sim D, a\sim\pi(s)}\left[Q_{\mathrm{LCB}}(s,a)\right] = \mathbb{E}_{s\sim D, a\sim\pi(s)}\left[\mathbb{E}_{\text{ens}}[Q_{\theta^i}(s,a)] + \beta\sqrt{\mathbb{V}_{\text{ens}}[Q_{\theta^i}(s,a)]}\right] \tag{4}$$

where $\beta \leq 0$ is a hyperparameter that determines the amount of pessimism.

4.2 The Trade-Off Between Trust and Pessimism

While our hope is to leverage the implicit generalization capabilities of neural networks to estimate proper LCBs beyond states and actions in the finite dataset D, neural network architectures can be fundamentally biased, or we can simply be in a setting with insufficient data coverage, such that the generalization capability of those networks is limited. To this end, we augment the policy evaluation objective of MSG ($\mathcal{L}(\theta^i)$, equation 3) with a support constraint regularizer inspired by CQL [11] [2]: $\mathcal{H}(\theta^i) = \mathbb{E}_{s\sim D, a\sim\pi(s)}\left[Q_{\theta^i}(s,a)\right] - \mathbb{E}_{(s,a)\sim D}\left[Q_{\theta^i}(s,a)\right]$. This regularizer encourages the Q-functions to increase the values for actions seen in the dataset D, while decreasing the values of the actions of the current policy. Practically, we estimate the latter expectation of $\mathcal{H}$ using the states in the mini-batch, and we approximate the former expectation using a single sample from the policy. We control the contribution of $\mathcal{H}(\theta^i)$ by weighting this term with weight parameter α. The full critic loss is thus given by,

$$\mathcal{L}(\theta^1, \ldots, \theta^N) = \sum_{i=1}^{N}\big(\mathcal{L}(\theta^i) + \alpha\mathcal{H}(\theta^i)\big) \tag{5}$$

Empirically, as evidenced by our results in Appendix A.2, we have observed that such a regularizer can be necessary in two situations: 1) The first scenario is where the offline dataset only contains a narrow data distribution (e.g., imitation learning datasets only containing expert data). We believe

[2] Instead of a CQL-style value regularizer, other forms of support constraints such as a behavioral cloning regularizer on the policy could potentially be used.

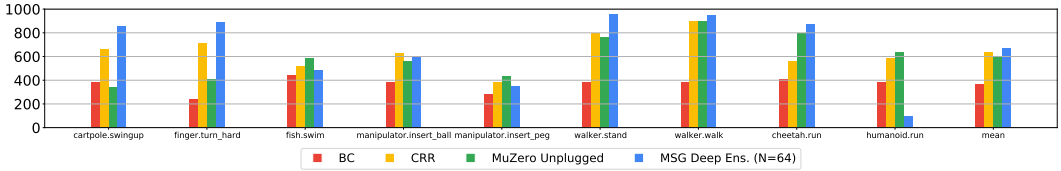


Figure 1: Results for DM Control Suite subset of the RL Unplugged benchmark [25]. We note that: 1) the architecture used for MSG is smaller by a factor of approximately 60x which contributes to poor performance on `humanoid.run`, 2) CRR results are reported by their best checkpoint throughout training which differs from MSG, BC, and MuZero Unplugged which report performance at the end of training. Baseline results taken from [47]. Despite MSG's disadvantage on the `humanoid.run` task, it still edges out the baseline methods in mean performance.

Domain	CQL	IQL	MSG ($N = 64$)	β	α	MSG ($N = 4$)	β	α
maze2d-umaze-v1	5.7	–	$\mathbf{101.1 \pm 26.3}$	-8	0	68.8 ± 20.2	-4	0.1
maze2d-medium-v1	5.0	–	$\mathbf{57.0 \pm 17.2}$	-4	0.1	$\mathbf{53.2 \pm 26.8}$	-4	0.1
maze2d-large-v1	12.5	–	$\mathbf{159.3 \pm 49.4}$	-4	0.1	59.2 ± 59.1	-8	0.5
antmaze-umaze-v0	74.0	87.5	$\mathbf{97.8 \pm 1.2}$	-4	0.5	$\mathbf{98.6 \pm 1.4}$	-4	1.0
antmaze-umaze-diverse-v0	**84.0**	62.2	$\mathbf{81.8 \pm 3.0}$	-4	1.0	76.6 ± 7.6	-4	0.5
antmaze-medium-play-v0	61.2	71.2	$\mathbf{89.6 \pm 2.2}$	-4	0.5	$\mathbf{83.0 \pm 7.1}$	-4	0.1
antmaze-medium-diverse-v0	53.7	70.0	$\mathbf{88.6 \pm 2.6}$	-4	0.5	$\mathbf{83.0 \pm 6.2}$	-4	0.5
antmaze-large-play-v0	15.8	39.6	$\mathbf{72.6 \pm 7.0}$	-8	0	46.8 ± 14.7	-4	0.5
antmaze-large-diverse-v0	14.9	47.5	$\mathbf{71.4 \pm 12.2}$	-8	0.1	58.2 ± 9.6	-8	0.1

Table 1: Results on D4RL maze2d and antmaze domains. In MSG, β is the hyperparameter controlling the amount of pessimism in Q_{LCB} (Equation 4), and α is the hyperparameter controlling the contribution of the CQL-style regularizer (Equation 5). As we were unable to reproduce CQL antmaze results despite extensive hyperparameter tuning (see also [12]), we present the numbers reported by the original paper which uses the same network architectures as MSG. We also present reported results for the current state-of-the-art, IQL [48].

this is because the power of ensembles comes from predicting a value distribution for unseen (s, a) based on the available training data. Thus, if no data for sub-optimal actions is present, ensembles cannot make accurate predictions and increased pessimism via $\mathcal{H}$ becomes necessary. 2) The second scenario is where environment dynamics can be chaotic (e.g. Gym [46] `hopper` and `walker2d`). In such domains it would be beneficial to remain close to the observed data in the offline dataset. Pseudo-code for our proposed MSG algorithm can be viewed in Algorithm Box 1.

5 Experiments

In this section we seek to empirically answer the following questions: 1) How well does MSG perform compared to current state-of-the-art in offline RL? 2) Are the theoretical differences in ensembling approaches (Section 3) practically relevant? 3) When and how does ensemble size affect perfomance? 4) Can we match the performance of MSG through efficient ensemble approximations developed in the supervised learning literature?

5.1 Offline RL Benchmarks

D4RL Gym Domains We begin by evaluating MSG on the Gym domains (`halfcheetah`, `hopper`, `walker2d`) of the D4RL offline RL benchmark [24], using the `medium`, `medium-replay`, `medium-expert`, and `expert` data settings. Our results presented in Appendix A.2 (summarized in Figure 4) demonstrates that MSG is competitive with well-tuned state-of-the-art methods CQL [11] and F-BRC [12].

D4RL Antmaze Domains Due to the narrow range of behaviors in Gym environments, offline datasets for these domains tend to be very similar to imitation learning datasets. As a result, many prior offline RL approaches that perform well on D4RL Gym fail on harder tasks that require stitching trajectories through dynamic programming (c.f. [48]). An example of such tasks are the D4RL antmaze settings, in particular those in the `antmaze-medium` and `antmaze-large` environments. The data for antmaze tasks consists of many episodes of an Ant agent [46] running along arbitrary

paths in a maze. The agent is tasked with using this data to learn a point-to-point navigation policy from one corner of the maze to the opposite corner, where rewards are given by a sparse signal that is 1 when near the desired end location in the maze – at which point the episode is terminated – and 0 otherwise. The undirected, extremely sparse reward nature of antmaze tasks make them very challenging, especially for the large maze sizes.

Table 1 and Appendix B.2 present our results. To the best of our knowledge, the antmaze domains are considered unsolved, with few prior works reporting non-zero results on the large mazes [11, 48]. As can be seen, MSG obtains results that far exceed the prior state-of-the-art results reported by [48]. While some works that use specialized hierarchical approaches have reported strong results as well [49], it is notable that MSG is able to solve these challenging tasks with standard architectures and training procedures, and this shows the power that ensembling can provide – as long as the ensembling is performed properly!

RL Unplugged In addition to the D4RL benchmark, we evaluate MSG on the RL Unplugged benchmark [25]. Our results are presented in Figure 1. We compare to results for Behavioral Cloning (BC) and two state-of-the-art methods in these domains, Critic-Regularized Regression (CRR) [7] and MuZero Unplugged [47]. Due to computational constraints when using deep ensembles, we use the same network architectures as we used for D4RL experiments. The networks we use are approximately $\frac{1}{60}$-th the size of those used by the BC, CRR, and MuZero Unplugged baselines in terms of number of parameters. We observe that MSG is on par with or exceeds the current state-of-the-art on all tasks with the exception of `humanoid.run`, which appears to require the larger architectures used by the baseline methods. Experimental details can be found in Appendix C.

Benchmark Conclusion Prior work has demonstrated that many offline RL approaches that perform well on Gym domains, fail to succeed on much more challenging domains [48]. Our results demonstrate that through uncertainty estimation with deep ensembles, MSG is able to very significantly outperform prior work on very challenging benchmark domains such as the D4RL antmazes.

5.2 Ensemble Ablations

Independence in Ensembles Ablation In Section 3, through theoretical arguments and toy experiments we demonstrated the importance of training using "Independent" ensembles. Here, we seek to validate the significance of our theoretical findings using offline RL benchmarks, by comparing `Independent` targets (as in MSG), to `Shared-LCB` and `Shared-Min` targets. Our results are presented in Appendices A.3 and B.3, with a summary in Figures 3 and 4.

In the Gym domains (Appendix A.3), with ensemble size $N = 4$, `Shared-LCB` significantly underperforms MSG. In fact, not using ensembles at all ($N = 1$) outperforms `Shared-LCB`. With ensemble size $N = 4$, `Shared-Min` is on par with MSG. When the ensemble size is increased to $N = 64$ (Figure 7), we observe the performance of `Shared-Min` drops significantly on $7/12$ D4RL Gym settings. In constrast, the performance of MSG is stable and does not change.

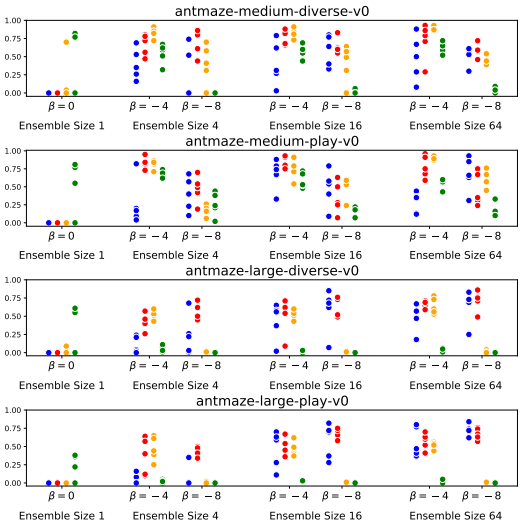


Figure 2: Ensemble size ablation on the antmaze medium and large domains with varying $\beta \in \{-4, -8\}$ and $\alpha \in \{0, 0.1, 0.5, 1\}$ (colored blue, red, yellow, and green respectively). We observe the general trend that bigger ensembles lead to better performance.

In the challenging antmaze domains (Appendix B.3), for both ensemble sizes $N = 4$ and $N = 64$, `Shared-LCB` and `Shared-Min` targets completely fail to solve the tasks, while for both ensemble sizes MSG exceeds the prior state-of-the-art (Table 1), IQL [48].

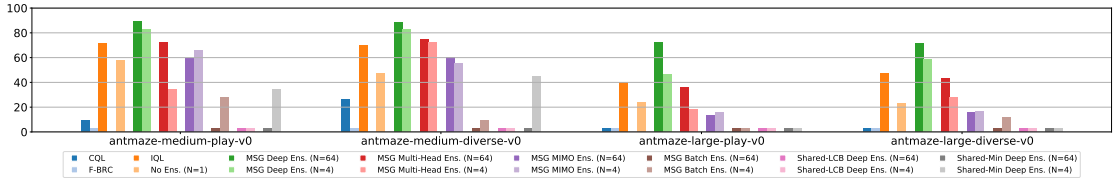


Figure 3: Summary of D4RL antmaze benchmark results (full results presented in Appendix B). For each method, we report the mean across random seeds for the best hyperparameter. Numerical results for all experiments are available in the supplementary material.

Independence in Ensembles Conclusion Our experiments corroborate the theoretical results in Section 3, demonstrating that `Independent` targets are critical to the success of MSG. These results are particularly striking when one considers that the implementations for MSG, `Shared-LCB`, and `Shared-Min` differ by only 2 lines of code.

Ensemble Size Ablation An important ablation is to understand the role of ensemble size in MSG.

In the Gym domains, Figure 5 demonstrates that increasing the number of ensembles from 4 to 64 does not result in a noticeable change in performance.

In the antmaze domains, we evaluate MSG under ensemble sizes $\{1, 4, 16, 64\}$. Figure 2 presents our results. Our key takeaways are as follows:

- For the harder `antmaze-large` tasks, there is a clear upward trend as ensemble size increases.
- Using a small ensemble size (e.g. $N = 4$) is already quite good, but more sensitive to hyperparameter choice especially on the harder tasks.
- Very small ensemble sizes benefit more from using $\alpha > 0$ [3]. However, across the board, using $\alpha = 0$ is preferable to using too large of a value for α – with the exception of $N = 1$ which cannot take advantange of the benefits of ensembling.
- When using lower values of β, lower values of α should be used.

Ensemble Size Conclusion In domains such as D4RL Gym where offline datasets are qualitatively similar to imitation learning datasets, larger ensembles do not result in noticeable gains. In domains such as D4RL antmaze which contain more data diversity, larger ensembles significantly improve the performance of agents.

5.3 Efficient Ensembles

Thus far we have demonstrated the significant performance gains attainable through MSG. An important concern however, is that of parameter and computational efficiency: Deep ensembles of Q-networks result in an N-fold increase in memory and compute usage, both in the policy evaluation and policy optimization phases of actor-critic training. While this might not be a significant problem in offline RL benchmark domains due to small model footprints[4], it becomes a major bottleneck with larger architectures such as those used in language and vision domains. To this end, we evaluate whether recent advances in "Efficient Ensemble" approaches from the supervised learning literature transfer well to the problem of offline RL. Specifically, the efficient ensemble approaches we consider are: Multi-Head Ensembles [26, 50, 51], MIMO Ensembles [27], and Batch Ensembles [28]. For a description of these efficient ensembling approaches please refer to Appendix E. A runtime comparison of different ensembling approaches can be viewed in Table 2.

D4RL Gym Domains Appendix A.4 presents our results in the D4RL Gym domains with ensemble size $N = 4$ (summary in Figure 4). Amongst the considered efficient ensemble approaches, Batch Ensembles [28] result in the best performance, which follows findings from the supervised learning literature [17].

[3] As a reminder, α is the weight of the CQL-style regularizer loss discussed in Section 4.2.

[4] All our experiments were ran on a single Nvidia P100 GPU.

D4RL Antmaze Domains Appendix B.4 presents our results in the D4RL antmaze domains for both ensemble sizes of $N = 4$ and $N = 64$ (summary in Figure 3). As can be seen, compared to MSG with deep ensembles (separate networks), the efficient ensemble approaches we consider are very unreliable, and fail for most hyperparameter choices.

Efficient Ensembles Conclusion We believe the observations in this section very clearly motivate future work in developing efficient uncertainty estimation approaches that are better suited to the domain of reinforcement learning. To facilitate this direction of research, in our codebase we have included a complete boilerplate example of an offline RL agent, amenable to drop-in implementation of novel uncertainty-estimation techniques.

6 Discussion & Future Work

Our work has highlighted the significant power of ensembling as a mechanism for uncertainty estimation for offline RL. In this work we took a renewed look into Q-ensembles, and studied how to leverage them as the primary source of pessimism for offline RL. Through theoretical analyses and toy constructions, we demonstrated a critical flaw in the popular approach of using shared targets for obtaining pessimistic Q-values, and demonstrated that it can in fact lead to optimistic estimates. Using a simple fix, we developed a practical deep offline RL algorithm, MSG, which resulted in large performance gains on established offline RL benchmarks.

As demonstrated by our experimental results, an important outstanding direction is to study how we can design improved efficient ensemble approximations, as we have demonstrated that current approaches used in supervised learning are not nearly as effective as MSG with ensembles that use separate networks. We hope that this work engenders new efforts from the community of neural network uncertainty estimation researchers towards developing efficient uncertainty estimation techniques directed at reinforcement learning.

Acknowledgments and Disclosure of Funding

We would like to thank Yasaman Bahri for insightful discussions regarding infinite-width neural networks. We would like to thank Laura Graesser for providing a detailed review of our work. We would like to thank conference reviewers for posing important questions that helped clarify the organization of this manuscript.

References

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

[2] 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.

[3] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[4] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[5] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

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

[7] Ziyu Wang, Alexander Novikov, Konrad Zolna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[8] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[9] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[10] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[11] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[12] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[13] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[14] Jacob Buckman, Carles Gelada, and Marc G Bellemare. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint arXiv:2009.06799*, 2020.

[15] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR, 2021.

[16] Tengyang Xie, Ching-An Cheng, Nan Jiang, Paul Mineiro, and Alekh Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *arXiv preprint arXiv:2106.06926*, 2021.

[17] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

[18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[19] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[20] Jordi Smit, Canmanie T Ponnambalam, Matthijs TJ Spaan, and Frans A Oliehoek. Pebl: Pessimistic ensembles for offline deep reinforcement learning. In *Robust and Reliable Autonomy in the Wild Workshop at the 30th International Joint Conference of Artificial Intelligence*, 2021.

[21] 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, 2021.

[22] 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*, pages 6131–6141. PMLR, 2021.

[23] 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*, pages 1702–1712. PMLR, 2022.

[24] 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.

[25] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A collection of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[26] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

[27] Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew M Dai, and Dustin Tran. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

[28] Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[29] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[30] P. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence off-policy evaluation. In *Proceedings of the 29th Conference on Artificial Intelligence*, 2015.

[31] Yihao Feng, Tongzheng Ren, Ziyang Tang, and Qiang Liu. Accountable off-policy evaluation with kernel bellman statistics. In *International Conference on Machine Learning*, pages 3102–3111. PMLR, 2020.

[32] Bo Dai, Ofir Nachum, Yinlam Chow, Lihong Li, Csaba Szepesvári, and Dale Schuurmans. Coindice: Off-policy confidence interval estimation. *arXiv preprint arXiv:2010.11652*, 2020.

[33] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11), 2009.

[34] Ilja Kuzborskij, Claire Vernade, Andras Gyorgy, and Csaba Szepesvári. Confident off-policy evaluation and selection through self-normalized importance weighting. In *International Conference on Artificial Intelligence and Statistics*, pages 640–648. PMLR, 2021.

[35] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[36] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*, 2016.

[37] Mengjiao Yang, Bo Dai, Ofir Nachum, George Tucker, and Dale Schuurmans. Offline policy selection under uncertainty. *arXiv preprint arXiv:2012.06919*, 2020.

[38] Ian Osband, Benjamin Van Roy, Daniel J Russo, Zheng Wen, et al. Deep exploration via randomized value functions. *J. Mach. Learn. Res.*, 20(124):1–62, 2019.

[39] Ilya Kostrikov and Ofir Nachum. Statistical bootstrapping for uncertainty estimation in off-policy evaluation. *arXiv preprint arXiv:2007.13609*, 2020.

[40] Ping Li, Jinde Cao, and Zidong Wang. Robust impulsive synchronization of coupled delayed neural networks with uncertainties. *Physica A: Statistical Mechanics and its Applications*, 373:261–272, 2007.

[41] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[42] Nick Pawlowski, Andrew Brock, Matthew CH Lee, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

[43] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[44] Shilin Zhu, Xin Dong, and Hao Su. Binary ensemble neural network: More bits per network or more networks per bit? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4923–4932, 2019.

[45] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[46] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[47] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. *arXiv preprint arXiv:2104.06294*, 2021.

[48] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[49] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[50] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29:4026–4034, 2016.

[51] Linh Tran, Bastiaan S Veeling, Kevin Roth, Jakub Swiatkowski, Joshua V Dillon, Jasper Snoek, Stephan Mandt, Tim Salimans, Sebastian Nowozin, and Rodolphe Jenatton. Hydra: Preserving ensemble diversity for model distillation. *arXiv preprint arXiv:2001.04694*, 2020.

[52] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32:8572–8583, 2019.

[53] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803*, 2019.

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

[55] Raphael Fonteneau, Susan A Murphy, Louis Wehenkel, and Damien Ernst. Batch mode reinforcement learning based on the synthesis of artificial trajectories. *Annals of operations research*, 208(1):383–416, 2013.

[56] Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020.

[57] Lantao Yu, Tianhe Yu, Chelsea Finn, and Stefano Ermon. Meta-inverse reinforcement learning with probabilistic context variables. *arXiv preprint arXiv:1909.09314*, 2019.

[58] Seyed Kamyar Seyed Ghasemipour, Shixiang Shane Gu, and Richard Zemel. Smile: Scalable meta inverse reinforcement learning through context-conditional policies. *Advances in Neural Information Processing Systems*, 32, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The abstract and introduction summarize the organization and contents of the paper.

(b) Did you describe the limitations of your work? [Yes] The most important limitation of using ensembles is their N-fold increase in computational cost. Section 5.3 frames this problem and investigates a variety of approaches used in the supervised learning literature. We demonstrate that while "Efficient Ensembles" perform well on the simpler D4RL Gym domains, they fail on the D4RL antmaze domains. We believe this section very clearly motivates future work in developing efficient uncertainty estimation approaches that are better suited to the domain of reinforcement learning.

(c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is focused on algorithmic advances in the field of offline reinforcement learning (RL). The potential positive and negative impacts of offline RL methods are directly function of the domain to which they are applied. In this work, we only apply our methods to open-sourced, simulated benchmark domains which do not have societal implications.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] Section 3.1 sets up the stage for the introduction of Theorem 3.1.

(b) Did you include complete proofs of all theoretical results? [Yes] Please refer to Appendix F.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Please refer to the supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The offline datasets are provided by the benchmarks we used. We have detailed all experimental details, with pointers to details in the respective sections.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In Table 1 we report mean and standard deviation across random seeds. For the summarized bar plots in Figures 3 and 4 we only plot the mean. This is because for each bar in this summary plot, we report the results for the best hyperparameter value (which for many settings, due to significant computational cost, had 2 random seeds). However, in this work we have shown in exact detail the dense hyperparameter searches we have done, and furthermore, in we have reported the result for every single experiment individually (i.e. every combination of data setting, method, hyperparameter, and random seed) (Figure 2, Appendix A, Appendix B). Reporting results for each individual experiment demonstrate the trends, even in the settings where each hyperparameter was ran with 2 seed. We have also included all of our numerical results as `.pkl` files in our codebase.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] We mentioned that all experiments were done on Nvidia P100 GPU devices. We did not include the total amount of compute we used.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite the D4RL [24] and RL Unplugged [25] benchmarks.

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include out codebase, python notebooks, as well as `.pkl` files of experimental results.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] D4RL [24] and RL Unplugged [25] are established and open-sourced offline RL datasets.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The D4RL [24] and RL Unplugged [25] do not contain such sensitive materials.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A D4RL Gym Locomotion Benchmarks

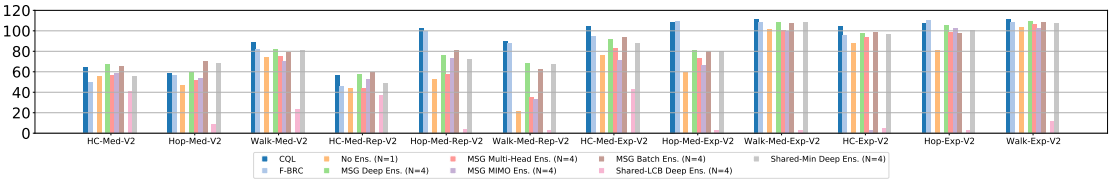


Figure 4: Summary of D4RL Gym benchmark results (full results presented in Appendix A). For each method, we report the mean across random seeds for the best hyperparameter. Numerical results for all experiments are available in the supplementary material.

In this section we discuss results using the D4RL gym (`halfcheetah`, `hopper`, `walker2d`) benchmark domains. The dataset types that we consider are `medium-v2`, `medium-replay-v2`, `medium-expert-v2`, and `expert-v2`.

A.1 Experimental Details

All policies and Q-functions are a 3 layer neural network with relu activations and hidden layer size 256. The policy output is a normal distribution that is squashed to $[-1, 1]$ using the tanh function. All methods were trained for 3M steps. CQL and MSG are trained with behavioral cloning (BC) for the first 50K steps. F-BRC pretrains a behavioral cloning model for 1M steps.

MSG, CQL, and F-BRC, are tuned with an equal hyperparameter search budget of 12 hyperparameter choices. Each hyperparameter choice for each method is trained using 2 random seeds. Each run is evaluated for 100 episodes. We report results for all hyperparameters and all seeds. For fairness of comparison, F-BRC is ran without adding a survival reward bonus. MSG and CQL are implemented in our code, and for F-BRC we use the opensourced codebase. Our reported CQL results appear to be better than or on par with values reported in prior works, which provided us with confidence to use our own implementation. We used the following values for hyperparameter search:

MSG $\beta \in \{-4. - 8.\}, \alpha \in \{0., 0.5, 1., 2., 4., 8.\}$

CQL $\alpha \in \{0., 0.45, 0.9, 1.36, 1.81, 2.27, 2.32, 3.18, 3.63, 4.09, 4.55, 5.\}$

F-BRC $\lambda \in \{0., 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\}$

A.2 Baseline Comparison

We compare MSG with deep ensembles to CQL [11] and F-BRC[12], two methods that obtain state-of-the-art results on the gym domains of the D4RL benchmark. Figure 5 presents our results on the gym domains. The key takeaways of our results are as follows:

- On gym domains, MSG is generally competitive with current state-of-the-art results.
- In the `hopper` and `walker2d` domains – where dynamics can be chaotic and staying close to the data-support is beneficial – CQL and F-BRC which rely on support constraints are more reliable w.r.t. the hyperparameter choice.

A.3 Ablations

We perform ablations w.r.t. the key components of MSG. Our key takeaways from the results in Figures 6 and 7 for gym domains are as follows:

- Comparing MSG – which uses independent targets – to Shared-LCB and Shared-Min – which use shared targets – clearly demonstrates the significance of our theoretical analysis in Section 3. Despite differing in only 2 lines of code from MSG, Shared-LCB and Shared-Min very significantly underpeform MSG. In fact, using no ensembling at all $(N = 1)$ outperforms Shared-LCB and is on par with Shared-Min.

- In the gym domains of D4RL – which are close to being imitation learning datasets – using large ensemble sizes does not result in noticeable gains in MSG.

A.4 Efficient Ensembles

As discussed in Section 5.3, we evaluate whether the performance of MSG with deep ensembles can be matched using state-of-the-art "efficient ensembles" from supervised learning literature. Our takeaways from the results in Figure 8 for gym domains are as follows:

- From the efficient ensembling approaches considered, Batch Ensembles tend to be the most performant [28]. Interestingly, this follows the findings of [17] from the supervised learning literature.

B D4RL Antmaze Benchmarks

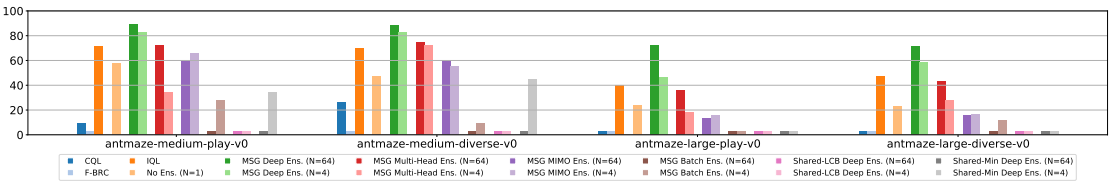


Figure 9: This Figure is identical to Figure 3. Summary of D4RL antmaze benchmark results (full results presented in Appendix B). For each method, we report the mean across random seeds for the best hyperparameter. Numerical results for all experiments are available in the supplementary material.

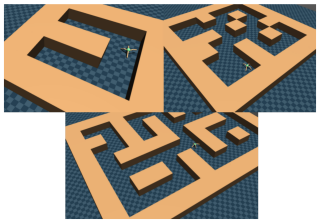


Figure 10: D4RL antmaze tasks. Figure taken from [24].

In this section we discuss results using the D4RL antmaze benchmark domains. We mainly focus our experiments on the `antmaze-medium` and `antmaze-large` environments, with the two available offline datasets of `play-v0` and `diverse-v0`.

B.1 Experimental Details

The rewards in the antmaze domains (which are either $0.$ or $1.$) were transformed using the following equation: $4(r - 0.5)$. This transformation has become common practice in prior works `https://github.com/aviralkumar2907/CQL/blob/d67dbe9cf5d2b96e3b462b6146f249b3d6569796/d4rl/examples/cql_antmaze_new.py#L22`.

All policies and Q-functions are a 3 layer neural network with relu activations and hidden layer size 256. The policy output is a normal distribution that is squashed to $[-1, 1]$ using the tanh function. All methods were trained for 2M steps. CQL and MSG are trained with behavioral cloning (BC) for the first 50K steps. F-BRC pretrains a behavioral cloning model for 1M steps.

MSG, CQL, and F-BRC, are tuned with an equal hyperparameter search budget of 8 hyperparameter choices. Each run is evaluated for 100 episodes. We report results for all hyperparameters and all seeds.

For all MSG with deep ensembles experiments (that appear in Figure 2 also), and for the No Ensembles ($N = 1$) baseline we ran 5 random seeds per hyperparameter choice. For all other baselines and ablations that appear in this section we ran 2 random seeds.

For fairness of comparison, F-BRC is ran without adding a survival reward bonus. MSG and CQL are implemented in our code, and for F-BRC we use the opensourced codebase. Our reported CQL results appear to be better than or on par with values reported in prior works, which provided us with confidence to use our own implementation. We used the following values for hyperparameter search:

MSG $\beta \in \{-4. - 8.\}, \alpha \in \{0., 0.1, 0.5, 1.\}$

CQL $\alpha \in \{0., 0.45, 0.9, 1.36, 1.81, 2.27, 2.32, 3.18, 3.63, 4.09, 4.55, 5.\}$

F-BRC $\lambda \in \{0., 0.14, 0.29, 0.43, 0.57, 0.71, 0.86, 1.\}$

B.2 Baseline Comparison

We compare MSG with deep ensembles to CQL [11] and F-BRC[12]. Figure 11 presents our results on the antmaze domains. The key takeaways of our results are as follows:

- On the D4RL antmaze domains, MSG with deep ensembles far exceeds the results of prior state of the art result Implicit Q-Learning (IQL) [48].
- CQL [11] and F-BRC [12] which obtain very strong results on the D4RL Gym domains (Appendix A.2), completely fail on the D4RL antmaze domains[5] This mimics the results of [48] that demonstrated many prior offline RL methods that perform strongly on D4RL Gym domains, fail on D4RL antmaze domains. We believe this highlights the importance of using more complex offline RL benchmarks that emphasize the stiching of diverse trajectories through dynamic programming, as opposed to offline RL datasets for Gym domains that are qualitatively very similar to imitation learning datasets.

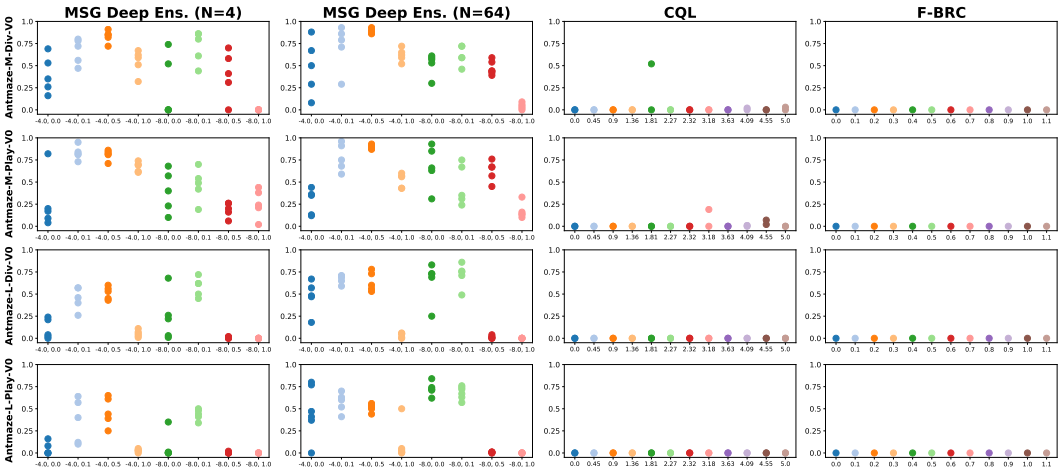


Figure 11: Basline comparison on D4RL antmaze domains. The hyperparameters for MSG, CQL, and F-BRC are (β, α), α, and λ respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

B.3 Ablations

We perform ablations w.r.t. the key components of MSG. Our key takeaways from the results in Figures 12 and 13 for antmaze domains are as follows:

[5]Note that our implementation of CQL exceeds the reported results of the original paper on D4RL Gym domains, providing us with confidence in our implementation. For D4RL antmaze domains we have also experimented with the open-sourced codebase accompanying [12], and the open-sourced codebase accompanying the original CQL work [11], but were unable to obtain the results reported in original paper.

- Comparing MSG – which uses independent targets – to Shared-LCB and Shared-Min – which use shared targets – clearly demonstrates the significance of our theoretical analysis in Section 3. Despite differing in only 2 lines of code from MSG, Shared-LCB and Shared-Min very significantly underpeform MSG. In fact, using no ensembling at all ($N = 1$) outperforms Shared-LCB and is on par with Shared-Min.

- In the gym domains of D4RL – which are close to being imitation learning datasets – using large ensemble sizes does not result in noticeable gains in MSG.

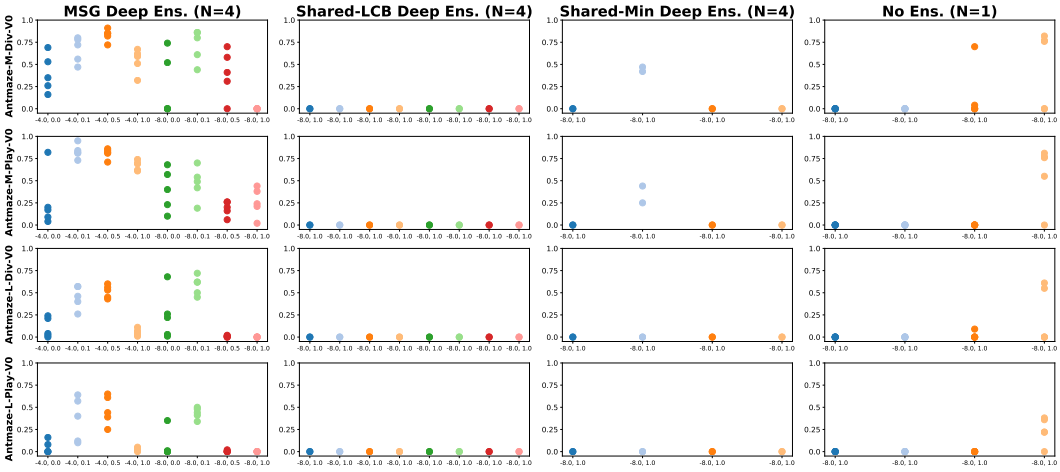


Figure 12: Ablations on D4RL antmaze domains ($N = 4$). The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

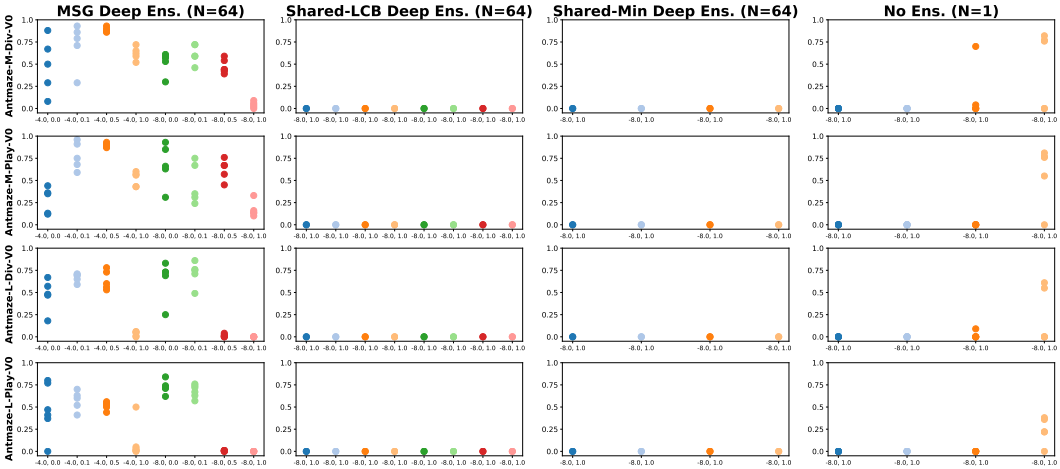


Figure 13: Ablations on D4RL antmaze domains ($N = 64$). The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

B.4 Efficient Ensembles

As discussed in Section 5.3, we evaluate whether the performance of MSG with deep ensembles can be matched using state-of-the-art "efficient ensembles" from supervised learning literature. Our takeaways from the results in Figures 14 and 15 for antmaze domains are as follows:

- Despite efficient ensembles such as Batch Ensembles performing well on the D4RL Gym domains (Appendix A.4), they fail on the antmaze domains for both ensemble sizes of $N = 4$ and $N = 64$.

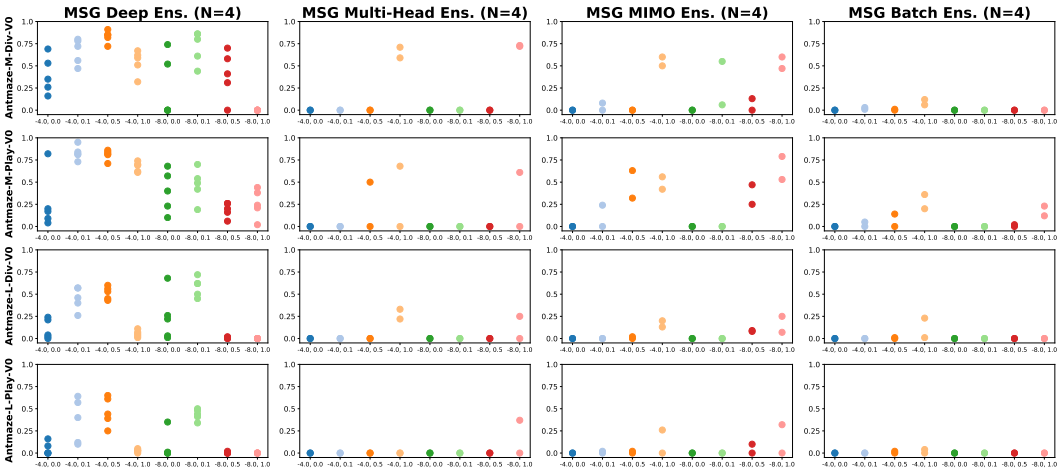


Figure 14: Efficient ensembles on D4RL antmaze domains ($N = 4$). The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

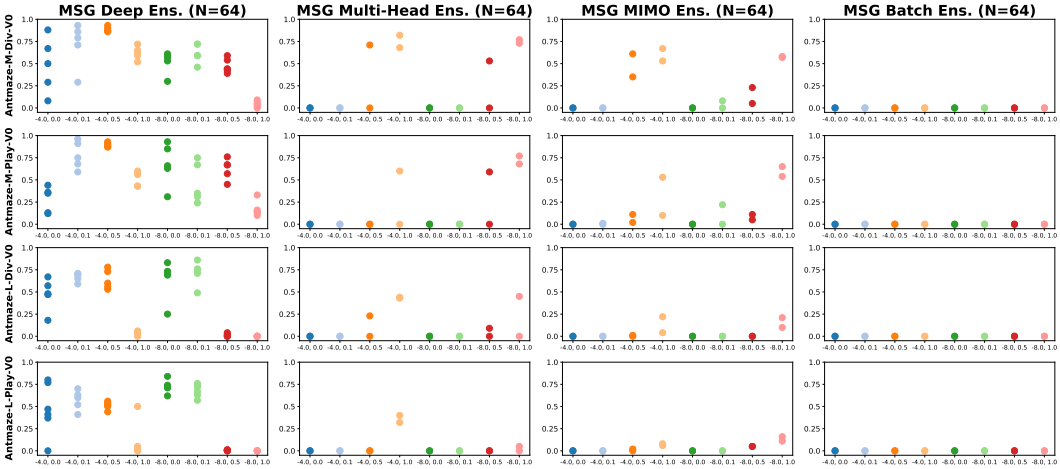


Figure 15: Efficient ensembles on D4RL antmaze domains ($N = 64$). The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

C RL Unplugged

C.1 DM Control Suite Tasks

The networks used in [25] for DM Control Suite Tasks are very large relative to the networks we used in the D4RL benchmark; roughly the networks contain 60x more parameters. Using a large ensemble size with such architectures requires training using a large number of devices. Furthermore, since in our experiments with efficient ensemble approximations we did not find a suitable alternative to deep ensembles (section 5.3), we decided to use the same network architectures and $N = 64$ as in the D4RL setting (enabling single-GPU training as before).

Our hyperparameter search procedure was similar to before, where we first performed a coarse search using 2 random seeds and hyperparameters $\beta \in \{-1., -2., -4. - 8.\}$, $\alpha \in \{0., 0.5\}$, and for the best found hyperparameter, ran final experiments with 5 new random seeds.

D Algorithm Box

Algorithm 1: Pseudocode for MSG Algorithm Using Deep Ensembles

Input: offline RL dataset D, Q-function and policy architectures, ensemble size N, MSG hyperparameters β, α

Create N Q-networks Q_{θ^i}, with parameters sampled from the initial weight distribution

Create target networks $Q_{\bar{\theta}^i}$, initialized as $\bar{\theta}^i \leftarrow \theta^i$

Create the policy network π with parameters θ^π

Function `PolicyEvaluationStep(`*batch*`):`

- **for** $i = 1, ..., N$ **do**
  - `/* Compute policy evaluation loss */`
  - $\forall s'_m \in batch, a'_{\pi,m} \sim \pi(s'_m)$
  - $L(\theta_i) = \frac{1}{M} \sum_m \Big(Q_{\theta^i}(s_m, a_m) - (r + \gamma \cdot Q_{\bar{\theta}^i}(s'_m, a'_{\pi,m})) \Big)^2$
  - `/* Compute support constraint regularizer loss */`
  - $\forall s_m \in batch, a_{\pi,m} \sim \pi(s'_m)$
  - $\mathcal{H}(\theta_i) = \frac{1}{M} \sum_m Q_{\theta^i}(s_m, a_{\pi,m}) - Q_{\theta^i}(s_m, a_m)$
  - `/* Optimize the policy evaluation objective */`
  - $\mathcal{L}(\theta^i) = L(\theta^i) + \alpha \cdot \mathcal{H}(\theta^i)$
  - $\theta^i \leftarrow \theta^i - \texttt{AdamUpdate}\Big(\mathcal{L}(\theta^i), \theta^i\Big)$
  - $\bar{\theta}^i \leftarrow \tau \cdot \bar{\theta}^i + (1 - \tau) \cdot \theta_i$

Function `MSGPolicyOptimizationStep(`*batch*`):`

- `/* Compute LCB` Q`-values */`
- $\forall s_m \in batch, a_{\pi,m} \sim \pi(s_m)$
- $\forall s_m \in batch, Q_{\text{LCB}}(s_m, a_{\pi,m}) = \underset{i}{\text{mean}} \left[Q_{\theta^i}(s_m, a_{\pi,m}) \right] - \beta \cdot \underset{i}{\text{std}} \left[Q_{\theta^i}(s_m, a_{\pi,m}) \right]$
- `/* Optimize the policy objective */`
- $\mathcal{L}(\theta^\pi) = \frac{1}{M} \sum_m Q_{\text{LCB}}(s_m, a_{\pi,m})$
- $\theta^\pi \leftarrow \theta^\pi - \texttt{AdamUpdate}\Big(\mathcal{L}(\theta^\pi), \theta^\pi\Big)$

Function `Main():`

- `/* Optional initialization of the policy using behavioral cloning while training` Q`-functions */`
- **for** *desired number of steps* **do**
  - batch $\sim D$
  - `PolicyEvaluationStep`(batch)
  - Update π using behavioral cloning
- **for** *desired number of steps* **do**
  - batch $\sim D$
  - `PolicyEvaluationStep`(batch)
  - `MSGPolicyOptimizationStep`(batch)

E Efficient Ensemble Approaches

Multi-Head [26, 50, 51] Multi-Head refers to ensembles that share a "trunk" network and have separate "head" networks for each ensemble member. In this work, we modify the last layer of a Q-network to output N predictions instead of a single Q-value, making the computational cost of this ensemble on par with a single network.

Multi-Input Multi-Output (MIMO) [27] MIMO is an ensembling approach that approximately has the same parameter and computational footprint as a single network. The MIMO approach only modifies the input and output layers of a given network. In MIMO, to compute predictions for a data-point x under an ensemble of size N, x is copied N times into $x_1, ..., x_N$, which are then

concatenated and passed to the network. The output of the network is a vector of size N, which is split into N predictions $y_1, ..., y_N$ representing the predictions of the ensemble members. For added clarification, we include Figure 16 depicting how a MIMO ensemble network functions. For further details on how MIMO networks are trained, we refer the interested reader to the original work of [27].

Batch Ensembles [28] Batch Ensembles incorporate rank-1 modulations to the weights of fully-connected layers. More specifically, let W be the weight matrix of a given fully-connected layer, and let x be the input to the layer. The output of the layer for ensemble member i is computed as $\sigma(((W^T(x \circ r^i)) \circ s^i) + b^i)$, where $\circ$ is the element-wise product, parameters with superscript i are separate for each ensemble member, and σ is the activation function. While Batch Ensembles is efficient in terms of number of parameters, in our actor-critic setup its computational cost is similar to deep ensembles, since for policy updates we need to evaluate every ensemble member using separate forward passes.

F Theorem Proof

In this section, following our notation and inifite-width networks setup described in Section 3.1, we present the proof for the following Theorem.

Theorem F.1. *For a given $(s, a) \in \mathcal{S} \times \mathcal{A}$, let $Q^{(0)}_{\theta^i}(s, a)$ denote $Q_{\theta^i}(s, a)|_{t=0}$ (value at initialization), with θ sampled from the initial weight distribution. After $t + 1$ iterations of pessimistic policy evaluation, the LCB value estimate for $(s', \pi(s')) \in \mathcal{X}'$ is given by,*

Independent Targets (Method 1):

$$Q^{(t+1)}_{\text{LCB}}(\mathcal{X}') = \mathcal{O}(\gamma^t \|C\|^t) + (1 + \ldots + \gamma^t C^t)CR - \sqrt{\mathbb{E}_{\text{ens}}\left[\left((1 + \ldots + \gamma^t C^t)(Q^{(0)}_{\theta^i}(\mathcal{X}') - CQ^{(0)}_{\theta^i}(\mathcal{X}))\right)^2\right]} \tag{6}$$

Shared Targets (Method 2):

$$Q^{(t+1)}_{\text{LCB}}(\mathcal{X}') = \mathcal{O}(\gamma^t \|C\|^t) + (1 + \ldots + \gamma^t C^t)CR - (1 + \ldots + \gamma^t C^t)\sqrt{\mathbb{E}_{\text{ens}}\left[\left(Q^{(0)}_{\theta^i}(\mathcal{X}') - CQ^{(0)}_{\theta^i}(\mathcal{X})\right)^2\right]} \tag{7}$$

where the square and square-root operations are applied element-wise. [6]

Proof. First we will establish some additional notation.

Consider an infinite-width NTK-parameterized [45] Q-function and let $Q^{(t)}(s, a)$ denote its value prediction for state-action pair (s, a) after t iterations of pessimistic policy evaluation. We will use the notation $\theta^{(t)}$ to refer to the parameters of this network after t iterations, but for simplicity of notation, when the context is clear we will omit the use of $\theta^{(t)}$. Additionally, consider the linearization (Taylor expansion) of Q about its initialization:

$$Q_{\text{lin}}(s, a) := Q^{(0)}(s, a) + \nabla_\theta Q^{(0)}(s, a) \cdot (\theta_{\text{lin}} - \theta^{(0)}) \tag{8}$$

When performing pessimistic policy evaluation using linearized networks, we will use $Q^{(t)}_{\text{lin}}(s, a)$ to denote the value prediction of Q_{lin} after t iterations, and will use $\theta^{(t)}_{\text{lin}}$ to refer to its learned weights. Note that $\theta^{(0)}_{\text{lin}} = \theta^{(0)}$, and that $\forall (s, a), Q^{(0)}(s, a) = Q^{(0)}_{\text{lin}}(s, a)$.

In [52] (section 2.4) it is shown that when training an infinitely wide neural network to perform regression using mean squared error, subject to technical conditions on the learning rate used, the predictions of the trained network are equivalent to if we had trained the linearized network instead. This means that after t iterations of our policy evaluation procedure, $\forall (s, a), t, Q^{(t)}_{\text{lin}}(s, a) = Q^{(t)}(s, a)$. *Hereafter we study the evolution of ensembles of linearized infinite-width networks, Q_{lin}, across training iterations.*

[6] Note that if $\gamma\|C\| \geq 1$, policy evaluation is liable to diverge in either setting. In our discussions, we avoid this degenerate case and assume $\gamma\|C\| < 1$.

As a reminder, $\mathcal{X}, R, \mathcal{X}'$ denote data matrices containing (s, a), r, and $(s', \pi(s'))$ appearing in the offline dataset D; i.e., the k-th transition (s, a, r, s') in D is represented by the k-th rows in $\mathcal{X}, R, \mathcal{X}'$. Additionally, we defined $C := \hat{\Theta}^{(0)}(\mathcal{X}', \mathcal{X}) \cdot \hat{\Theta}^{(0)}(\mathcal{X}, \mathcal{X})^{-1}$. Below, we will use notation such as $Q(\mathcal{X})$ to mean applying Q to each row of $\mathcal{X}$ and stacking the predictions into a vector. By $\hat{\Theta}^{(0)}((s, a), \mathcal{X})$ we mean to treat (s, a) as a row matrix and compute the $1 \times |D|$ kernel matrix.

Derivation for Independent Targets When using Independent Targets, in iteration $t + 1$ each network uses its own temporal difference (TD) targets,

$$\mathcal{Y}^{(t)} = R + \gamma Q^{(t)}_{\text{lin}}(\mathcal{X}') \tag{9}$$

Using the equations in [52] (section 2.2, equations 9-10-11) we can derive,

$$Q^{(t+1)}_{\text{lin}}(\mathcal{X}) = \mathcal{Y}^{(t)} \tag{10}$$

$$\forall s, a, Q^{(t+1)}_{\text{lin}}(s, a) = Q^{(0)}(s, a) + \hat{\Theta}^{(0)}\left((s, a), \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\right) \tag{11}$$

$$Q^{(t+1)}_{\text{lin}}(\mathcal{X}') = Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}', \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\right) \tag{12}$$

Plugging in the expression for the targets $\mathcal{Y}^{(t)}$ and recursively expanding the expressions above we obtain,

$$\begin{aligned}
Q^{(t+1)}_{\text{lin}}(\mathcal{X}') &= Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}', \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\right) && (13)\\
&= Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}', \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(R + \gamma Q^{(t)}_{\text{lin}}(\mathcal{X}') - Q^{(0)}(\mathcal{X})\right) && (14)\\
&= Q^{(0)}(\mathcal{X}') + C \cdot \left(R + \gamma Q^{(t)}_{\text{lin}}(\mathcal{X}') - Q^{(0)}(\mathcal{X})\right) && (15)\\
&= Q^{(0)}(\mathcal{X}') + CR - CQ^{(0)}(\mathcal{X}) + \gamma C Q^{(t)}_{\text{lin}}(\mathcal{X}') && (16)\\
&= \dots && (17)\\
&= (1 + \ldots + \gamma^t C^t)\left(Q^{(0)}(\mathcal{X}') + CR - CQ^{(0)}(\mathcal{X})\right) + (\gamma C)^{t+1} Q^{(0)}(\mathcal{X}') && (18)
\end{aligned}$$

Per our earlier discussion above, from Equation 18 we can conclude that,

$$Q^{(t+1)}(\mathcal{X}') = Q^{(t+1)}_{\text{lin}}(\mathcal{X}') = (1 + \ldots + \gamma^t C^t)\left(Q^{(0)}(\mathcal{X}') + CR - CQ^{(0)}(\mathcal{X})\right) + (\gamma C)^{t+1} Q^{(0)}(\mathcal{X}') \tag{19}$$

We can now derive Q_{LCB} for the Independent Targets setting, by computing the expectation and variance of $Q^{(t+1)}(\mathcal{X}')$ with respect to the initial weight distribution. Following [52] we obtain,

$$\mathbb{E}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')] = (1 + \ldots + \gamma^t C^t) CR \tag{20}$$

$$\mathbb{V}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')] = \mathbb{E}_{\text{ens}}\left[\left(\left(1 + \ldots + \gamma^t C^t\right)\left(Q^{(0)}(\mathcal{X}') - CQ^{(0)}(\mathcal{X})\right) + (\gamma C)^{t+1} Q^{(0)}(\mathcal{X}')\right)^2\right] \tag{21}$$

$$= \mathbb{E}_{\text{ens}}\left[\left(\left(1 + \ldots + \gamma^t C^t\right)\left(Q^{(0)}(\mathcal{X}') - CQ^{(0)}(\mathcal{X})\right)\right)^2\right] + \mathcal{O}(\gamma^t \|C\|^t) \tag{22}$$

We thus have,

$$\begin{aligned}
Q^{(t+1)}_{\text{LCB}}(\mathcal{X}') &= \mathbb{E}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')] - \sqrt{\mathbb{V}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')]} && (23)\\
&= \mathcal{O}(\gamma^t \|C\|^t) + (1 + \ldots + \gamma^t C^t) CR\\
&\quad - \sqrt{\mathbb{E}_{\text{ens}}\left[\left((1 + \ldots + \gamma^t C^t)(Q^{(0)}_{\theta^i}(\mathcal{X}') - CQ^{(0)}_{\theta^i}(\mathcal{X}))\right)^2\right]}
\end{aligned}$$

where the square and square-root operators are applied element-wise. Note that if $\gamma\|C\| \geq 1$, policy evaluation is liable to diverge. Thus, in our discussions we avoid this degenerate case and assume $\gamma\|C\| < 1$, which makes the term $\mathcal{O}(\gamma^t \|C\|^t)$ negligible as $t \to \infty$.

Derivation for Shared LCB Targets When using Shared LCB Targets, in iteration $t+1$ each network uses pessimistic temporal difference (TD) targets that are shared amongst ensemble members,

$$\mathcal{Y}^{(t)} = \text{LCB}\Big(R + \gamma Q_{\text{lin}}^{(t)}(\mathcal{X}')\Big) \tag{24}$$

$$= R + \text{LCB}\Big(\gamma Q_{\text{lin}}^{(t)}(\mathcal{X}')\Big) \tag{25}$$

Using the equations in [52] (section 2.2, equations 9-10-11) we can derive,

$$Q_{\text{lin}}^{(t+1)}(\mathcal{X}) = \mathcal{Y}^{(t)} \tag{26}$$

$$\forall s, a, Q_{\text{lin}}^{(t+1)}(s,a) = Q^{(0)}(s,a) + \hat{\Theta}^{(0)}\left((s,a),\mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X},\mathcal{X}\right)^{-1} \cdot \Big(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\Big) \tag{27}$$

$$Q_{\text{lin}}^{(t+1)}(\mathcal{X}') = Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}',\mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X},\mathcal{X}\right)^{-1} \cdot \Big(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\Big) \tag{28}$$

$$= Q^{(0)}(\mathcal{X}') + C \cdot \Big(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\Big) \tag{29}$$

Noting that in the Shared-LCB setting $\mathcal{Y}^{(t)}$ is not a random variable, we can now compute the expectation and variance of $Q_{\text{lin}}^{(t+1)}(\mathcal{X}')$ with respect to the initial weight distribution,

$$\mathbb{E}_{\text{ens}}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')] = \mathbb{E}_{\text{ens}}\Big[Q^{(0)}(\mathcal{X}') + C \cdot \Big(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\Big)\Big] = C\mathcal{Y}^{(t)} \tag{30}$$

$$\mathbb{V}_{\text{ens}}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')] = \mathbb{E}_{\text{ens}}\Big[\Big(Q^{(0)}(\mathcal{X}') - CQ^{(0)}(\mathcal{X})\Big)^2\Big] = \text{same } \forall t+1 \geq 1 \tag{31}$$

Let $A := \sqrt{\mathbb{V}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')]}$. Given the above equations, we can recursively compute the closed form for $\mathcal{Y}^{(t)}$ as follows,

$$\mathcal{Y}^{(t)} = R + \text{LCB}\Big(\gamma Q_{\text{lin}}^{(t)}(\mathcal{X}')\Big) \tag{32}$$

$$= R + \gamma \cdot \Big[\mathbb{E}[Q_{\text{lin}}^{(t)}(\mathcal{X}')] - \mathbb{V}[Q_{\text{lin}}^{(t)}(\mathcal{X}')]\Big] \tag{33}$$

$$= R + \gamma \cdot \Big[C\mathcal{Y}^{(t-1)} - A\Big] \tag{34}$$

$$= R - \gamma A + \gamma C\mathcal{Y}^{(t-1)} \tag{35}$$

$$= \dots \tag{36}$$

$$= (1 + \dots + \gamma^{t-1}C^{t-1})(R - \gamma A) + \gamma^t C^t \mathcal{Y}^{(0)} \tag{37}$$

$$= (1 + \dots + \gamma^{t-1}C^{t-1})(R - \gamma A) + \gamma^t C^t R + \gamma^{t+1}C^t \text{LCB}(Q^{(0)}(\mathcal{X}')) \tag{38}$$

Based on this form, we can write,

$$\mathbb{E}_{\text{ens}}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')] = C\mathcal{Y}^{(t)} \tag{39}$$

$$= (1 + \dots + \gamma^t C^t)CR - (\gamma C + \dots + \gamma^t C^t)A + \gamma^{t+1}C^{t+1}\text{LCB}(Q^{(0)}(\mathcal{X}')) \tag{40}$$

Combining our derivations, and noting from our earlier discussion that $\forall (s,a), t, Q_{\text{lin}}^{(t)}(s,a) = Q^{(t)}(s,a)$, we have,

$$Q_{\text{LCB}}^{(t+1)}(\mathcal{X}') = \mathbb{E}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')] - \sqrt{\mathbb{V}_{\text{ens}}[Q^{(t+1)}(\mathcal{X}')]} \tag{41}$$

$$= \mathbb{E}_{\text{ens}}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')] - \sqrt{\mathbb{V}_{\text{ens}}[Q_{\text{lin}}^{(t+1)}(\mathcal{X}')]} \tag{42}$$

$$= (1 + \dots + \gamma^t C^t)CR - (\gamma C + \dots + \gamma^t C^t)A + \gamma^{t+1}C^{t+1}\text{LCB}(Q^{(0)}(\mathcal{X}')) - A \tag{43}$$

$$= (1 + \dots + \gamma^t C^t)CR - (1 + \dots + \gamma^t C^t)A + \gamma^{t+1}C^{t+1}\text{LCB}(Q^{(0)}(\mathcal{X}')) \tag{44}$$

Thus, we have our results,

$$Q_{\text{LCB}}^{(t+1)}(\mathcal{X}') = \mathcal{O}(\gamma^t \|C\|^t) + (1 + \ldots + \gamma^t C^t) CR \\ - (1 + \ldots + \gamma^t C^t) \sqrt{\mathbb{E}_{\text{ens}}\left[\left(Q_{\theta^i}^{(0)}(\mathcal{X}') - C Q_{\theta^i}^{(0)}(\mathcal{X})\right)^2\right]} \tag{45}$$

where the square and square-root operators are applied element-wise. Note that if $\gamma\|C\| \geq 1$, policy evaluation is liable to diverge. Thus, in our discussions we avoid this degenerate case and assume $\gamma\|C\| < 1$, which makes the term $\mathcal{O}(\gamma^t\|C\|^t)$ negligible as $t \to \infty$.

The derivations of Equations 23 and 45 conclude our proof. □

G A Pedagogical Toy MDP

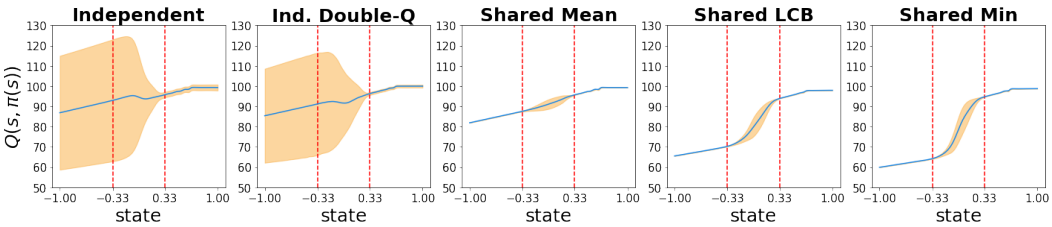


Figure 17: Verifying theoretical predictions on the toy Continuous Chain MDP. The marked interval $[-0.33, 0.33]$ denotes the region of state-space with no data. As anticipated by Theorem 3.1, when the Q-value functions are trained independently, the derived uncertainties capture the interaction between the available data, the structure of the MDP, and the policy being evaluated.

G.1 Qualitative Evaluation of Obtained Uncertainties

In this section we continue to study the implications of our theoretical results in Section 3 by constructing a pedagogical toy MDP. Our simple toy MDP allows us to follow the idealized setting of the presented theorem more closely, and allows for visualization of uncertainties obtained through different ensembling approaches.

Continuous Chain MDP The MDP we consider has state-space $\mathcal{S} = [-1, 1]$, action space $\mathcal{A} \in \mathbb{R}$, deterministic transition dynamics $s' = s + a$ clipped to remain inside $\mathcal{S}$, and reward function $r(s, a) = \mathbb{1}[s' \in [0.75, 1]]$.

Data Collection & Evaluation Policy The offline dataset we generate consists of 40 episodes, each of length 30. At the beginning of each episode we initialize at a random state $s \in \mathcal{S}$. In each step we take a random action sampled from $\text{Unif}(-0.3, 0.3)$, and record all transitions (s, a, r, s'). For evaluating the uncertainties obtained from different approaches, we create regions of missing data by removing all transitions where s or s' fall in the range $[-0.33, 0.33]$. The policy used for policy evaluation with the different ensembling approaches is $\forall s, \pi(s) = 0.1$.

Optimal Desired Form of Uncertainty Note that the evaluation policy $\pi(s) = 0.1$ is always moving towards the positive direction, and there is lack of data for states in the interval $[-0.33, 0.33]$. Hence, what we would expect is that in the region $[0.33, 1]$ there should not be a significant amount of uncertainty, while in the region $[-1, 0.33]$ there should be significantly more uncertainty about the Q-values of π, because the policy will be passing through $[-0.33, 0.33]$ where there is no data. Furthermore, as we move towards the negative axis, we would expect that in the region $[-0.33, 0.33]$ the uncertainty would gradually increase, while in the region $[-1, -0.33]$ the uncertainty would not continue to increase.

Results We visualize and compare the uncertainties obtained when the targets in the pessimistic policy evaluation procedure are computed as:

- **Independent Targets (as used in MSG):** $y^i = r + \gamma \cdot Q_{\theta^i}(s', \pi(s'))$
- **Independent Double-Q Targets:** $y^i = r + \gamma \cdot \min\Big[Q^1_{\theta^i}(s', \pi(s')), Q^2_{\theta^i}(s', \pi(s'))\Big]$
- **Shared Mean Targets:** $y = r + \gamma \cdot \text{mean}\Big[Q_{\theta^i}(s', \pi(s'))\Big]$
- **Shared LCB Targets:** $y = r + \gamma \cdot \left[\text{mean}\Big[Q_{\theta^i}(s', \pi(s'))\Big] - 2 \cdot \text{std}\Big[Q_{\theta^i}(s', \pi(s'))\Big]\right]$
- **Shared Min Targets:** $y = r + \gamma \cdot \min\Big[Q_{\theta^i}(s', \pi(s'))\Big]$

Note that `Independent Double-Q` is still an independent ensemble, where each ensemble member has an architecture containing a min-pooling on top of two subnetworks.

In Figure 17 we plot the mean and two standard deviations of the Q-values predicted by the ensemble for the policy we evaluated, $\pi(s) = 0.1$ (additional experimental details presented in Appendix G.3). The first striking observation is that, `Independent` targets (as used in MSG) effectively match our desired form of uncertainty: states that under the evaluation policy $\pi(s) = 0.1$ would lead to regions with little data have wider uncertainties than states that do not. A second observation is that, in this toy construction, `Shared LCB` and `Shared Min` provide a seemingly good approximation to the lower-bound of `Independent` predictions. However, our theoretical results show that shared targets have critical failure cases. Furthermore, our empirical results on challenging benchmark domains (Section 5.1) demonstrate that `Shared LCB` and `Shared Min` targets completely fail to train successful policies, despite their implementation differing from `Independent` targets in only 2 lines of code.

G.2 NTK vs. Maximal Parameterization

The toy experiment presented in section G.1 uses a single-hidden layer finite-width neural network architecture with `tanh` activations. The networks use the "standard weight parameterization" (i.e. the weight parameterization used in practice) as opposed to the NTK parameterization [53], and we optimize the networks using the Adam optimizer [54]. While this setup is close to the practical setting and demonstrates the relevance of independent ensembles for the practical setting, an important question posed by our reviewers is how close these results are to the theoretical predictions presented in 3.1. To answer this question, we present the following set of results.

Using the identical MDP and offline data as before, we implement 1 hidden layer neural networks with `erf` non-lineartiy. The networks are implemented using the Neural Tangents library [53], and use the NTK parameterization. The networks in the ensemble are optimized using full-batch gradient descent with learning rate 1 for 500 steps of Fitted Q-Evaluation (FQE) [55], where in each FQE step the networks are updated for 1000 gradient steps. We vary the width of the networks from 32 to 32768 in increments of a factor of 4, plotting the mean and standard deviation of the network predictions. The ensemble size is set to $N = 16$, except for width 32768 where $N = 4$.

We compare the results for finite-width networks to computing results for the infinite-width setting in closed form (using Theorem 3.1). Using the Neural Tangents library [53] we obtained the NTK for the architecture described in the previous paragraph (1 hidden layer with `erf` non-linearity). We found that the matrix inversion required in our equations results in numerical errors. Hence, we make the modification $\Theta(\mathcal{X}, \mathcal{X}) \leftarrow \Theta(\mathcal{X}, \mathcal{X}) + \texttt{1e-3} \cdot \text{I}$.

Figure 18 presents our results. As the width of the networks grow larger, the shape of the uncertainties becomes more similar to our closed-form equations (i.e. the variances become very small). While we do not have a rigorous explanation for why finite-width networks exhibit intuitively more desirable behaviors, we present below a strong hypothesis backed by empirical evidence. We believe rigorously answering this question is an incredibly interesting avenue for future work.

Hypothesis: Infinite-width networks in the NTK parameterization do not learn data-dependent features [56]. [56] present a different approach for parameterizing infinite-width networks called the "Maximal Parameterization", which enables inifinite-width networks to learn data-dependent features. We perform the same experiment as above, by replacing the NTK-parameterized networks with Maximal Parameterizations. Figure 18 presents our empirical results for network widths from

32 to 32768. Excitingly, we observe that with Maximial Parameterization, even our widest networks recover the intuitively desired form of uncertainty described in section G.1! The solutions of these networks also appear much more accurate, particularly on the right hand side of the plot; we can observe the correct stepped structure of the $Q - values$ up until the interval $[0.75, 1]$ where the policy always receives a reward of 1. Each step appears to be approximately 0.1 in width, which is size of the action of the policy being evaluated, $\forall s, \pi(s) = 0.1$.

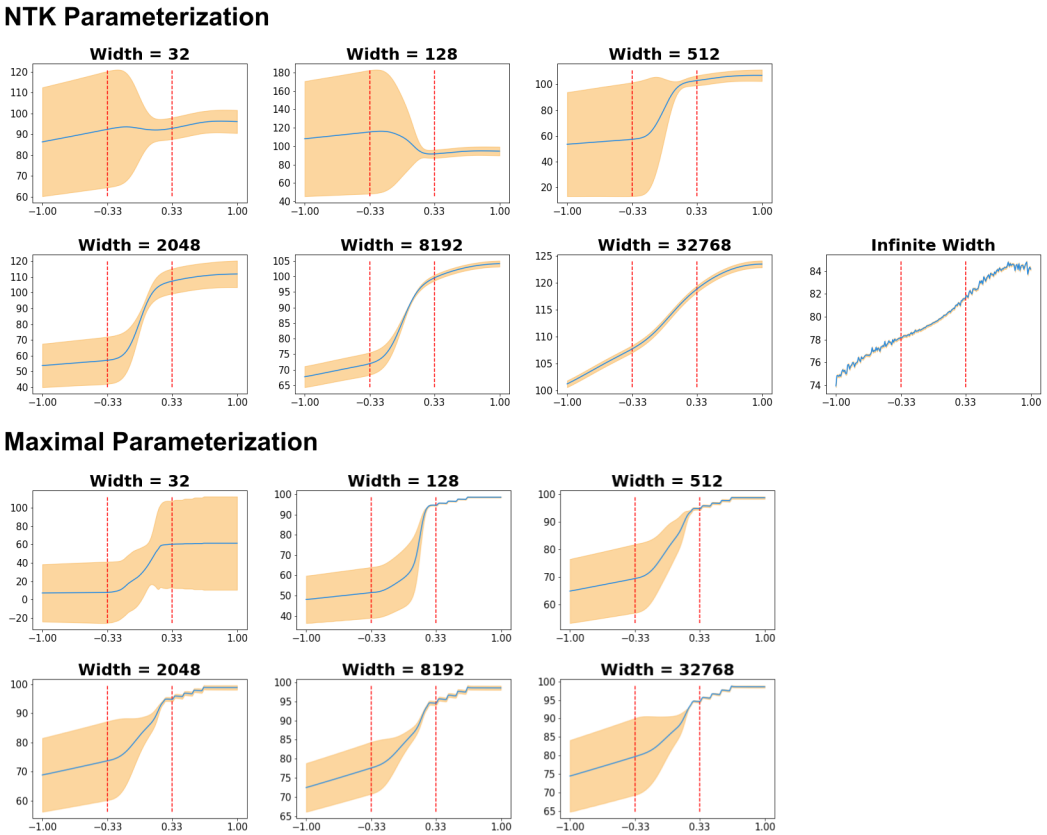


Figure 18: Comparing results of finite-width networks to closed form equations derived in Theorem 3.1. In the NTK parameterization, as width $\to \infty$, the structure of the variances collapse and resemble the infinite-width closed-form results. We believe this is due to infinite-width networks under the NTK regime not being able to learn features [56]. Supporting this hypothesis, we observe that networks parameterized by the Maximal Parameterization of [56] maintain the desired uncertainty structure as the width of the networks grows larger.

G.3 Additional Implementation Details for Figure 17

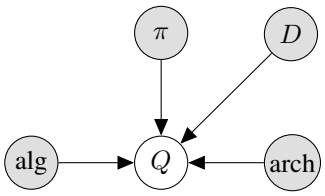
To evaluate the quality of uncertainties obtained from different Q-function ensembling approaches, we create $N = 64$ Q-function networks, each being a one hidden layer neural network with hidden dimension 512 and `tanh` activation. The initial weight distribution is a fan-in truncated normal distribution with scale 10.0, and the initial bias distribution is fan-in truncated normal distribution with scale 0.05. We did not find results with other activation functions and choices of initial weight and bias distribution to be qualitatively different. We use discount $\gamma = 0.99$ and the networks are optimized using the Adam [54] optimizer with learning rate `1e-4`. In each iteration, we first compute the TD targets using the desired approach (e.g. independent vs. shared targets) and then fit the Q-functions to their respective targets with 2000 steps of full batch gradient descent. We train the networks for 1000 such iterations (for a total of 2000×1000 gradient steps). Note that we do not use target networks. Given the small size of networks and data, these experiments can be done within a few minutes using a single GPU in a `Google Colaboratory` notebook which we have included in the supplementary material.

H Practical Workflow for Applying MSG to New Offline RL Datasets

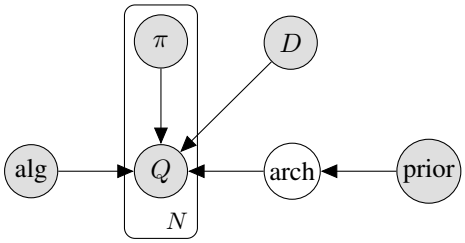
Throughout this work, we have reported the results for every method, domain, hyperparameter choice, and random seed. Based on our experience, in this section we outline a practical workflow for applying MSG to new offline RL environments and datasets.

In addition to the the key take-aways from our discussion of Figure 2 in Section 5.2, here we include practical advice on hyperparameter tuning when applying MSG to a new domain. Aside from the choice of neural network architectures, the main hyperparameters in MSG are β, i.e. the hyperparameter controlling the amount of pessimism in Q_{LCB} (Equation 4), and α, i.e. the hyperparameter controlling the contribution of the CQL-style regularizer (Equation 5). Intuitively, as described in Section 4.2, larger values of α become necessary when the offline dataset has a narrow distribution (e.g. imitation learning datasets), or in MDPs with more chaotic dynamics, where small deviations can be very costly and thus we must stay close to the provided data support. Equipped with this intuition, for a new offline RL task and dataset, we would first use low β values (e.g. $\beta \in \{-4, -8\}$ or lower), and search for an appropriate range of α (which may be 0, as in our reported antmaze-large results in Table 1). For further hyperparameter tuning from this starting point, we would investigate if reducing pessimism by increasing β – and potentially modifying α – would lead to improved policies. Generally, we find MSG to be fairly robust, as evidenced by our results in Appendices A and B, and Figure 2.

I Statistical Model



(a) Q-function generative process: the graphical model above represents the induced distribution over Q-functions when conditioning on a particular policy evaluation algorithm, policy, offline dataset, and Q-function network architecture.



(b) An example of an interesting extension

An interesting question posed by reviewers of our work was "[W]hatever formal reasoning system we'd like to use, what is the ideal answer, given access to arbitrary computational resources, so that approximations are unnecessary? I.e., how do we quantify our uncertainty about the MDP and value function before seeing data, and how do we quantify it after seeing data?"

It is important to begin by clarifying what is the mathematical object we are trying to obtain uncertainties over. In this work, we do not quantify uncertainties about any aspects of the MDP itself (although this is an interesting question which comes up in model-based methods as well as other settings such as Meta-Learning [57, 58]). Our goal in this work is to directly estimate $Q^\pi(s,a) = r(s,a) + \gamma \cdot \mathbb{E}_{s' \sim MDP, a' \sim \pi}[Q(s',a')]$, for $a \sim \pi(s)$, and to obtain uncertainties about $Q^\pi(s,a)$.

Let $Q(s,a)$ be a predictor $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that needs to be evaluated on – and hopefully generalize well to – $(s,a) \notin D$, where D is the offline dataset. When we choose to represent $Q(s,a)$ using neural networks, Gaussian Processes, or K-nearest-neighbours, we are not just making approximations for computational reasons, but are actually choosing a function class which we believe will generalize well to unseen (s,a).

One practical example of learning Q-functions is to use Fitted Q-Evaluation (FQE) [55] on the provided data using gradient descent with a desired neural network architecutre. Due to the random weight initialization, this procedure induces a distribution on the Q-functions which is captured by the probabilistic graphical model (PGM) in Figure 19a. In other words, by conditionining on the policy, data, architecture, and policy evaluation algorithm, we are imposing a belief over Q-functions. Note that this is effectively the same justification as using ensembles in supervised deep learning,

where ensembles are state-of-the-art for accuracy and calibration [17]. For the sake of theoretical analysis (Section 3), we studied this belief distribution under the infinite-width NTK [45, 52] network setting, in which case the distribution over Q-functions is a Gaussian Process.

The focus of this work is to ask the question: "Under this imposed belief, what should the policy update be?". Our proposed answer is to optimize the policy with respect to the lower-confidence bound of our beliefs: In the standard actor-critic setup, the policy optimization objective takes the form $\max_\pi \mathbb{E}_{d(s)}[Q(s, \pi(s))]$, where $d(s)$ is some distribution over states (e.g. the initial state distribution, a uniform distribution over states in the offline dataset D, etc.). For the pessimistic offline RL settting, our proposed policy optimization objective takes the form $\max_\pi \text{LCB}\Big(\mathbb{E}_{d(s)}[Q(s, \pi(s))]\Big)$. In MSG, for practical reasons, we convert this to $\max_\pi \mathbb{E}_{d(s)}\Big[\text{LCB}\Big(Q(s, \pi(s))\Big)\Big]$ (which is a lower-bound of the latter objective).

The graphical model in Figure 19b also highlights an example of interesting future directions: Consider an offline RL setup where we keep track of the various policies generating the data and their Q-functions (approximated through Monte-Carlo estimation). Then, we can impose a prior distribution over architectures, and using the available Q-functions, infer a posterior distribution over architectures (for a probabilistic interpretation, we could treat the output of a Q-function as the mean of a standard normal distribution). Subsequently, given a new policy, we can learn its Q-function under the posterior neural network architecture distribution.

J Runtime Comparison Table

Method	**Runtime**
No Ens. ($N = 1$) ($\alpha > 0$)	$1\times$
CQL	$0.27\times$
F-BRC	$0.22\times$
MSG Deep Ens. ($N = 4$)	$0.64\times$
MSG Multi-Head Ens. ($N = 4$)	$0.92\times$
MSG MIMO Ens. ($N = 4$)	$0.90\times$
MSG Batch Ens. ($N = 4$)	$0.72\times$
MSG Deep Ens. ($N = 64$)	$0.11\times$
MSG Multi-Head Ens. ($N = 64$)	$0.88\times$
MSG MIMO Ens. ($N = 64$)	$0.61\times$
MSG Batch Ens. ($N = 64$)	$0.29\times$

Table 2: Runtime comparison of various approaches. We report training iterations per second, relative to no ensembling while using the CQL-style regularizer described in Section 4.2. All methods were trained on an Nvidia P100 GPU.

K MSG-LCB vs. MSG-Min

In the deep offline RL literature, ensembles are typically employed using the Shared-Min Targets formulation [9, 3, 8, 4]. Under the NTK setting of Theorem 3.1, using the $\min$ formulation is mathematically infeasible, since the distribution of Q-values under the ensemble is Gaussian, leading to $\inf_{\text{ens}} Q(s, a) = -\infty$ for all (s, a). Additionally, using the LCB formulation provides the β hyperparameter as a knob for tuning the extent of pessimism. Nonetheless, in practical settings with finite ensemble sizes, using MSG with a $\min$ formulation is feasible. Figure 20 presents our empirical results comparing MSG-LCB and MSG-Min on the D4RL antmaze domains. We observe that on the antmaze-medium domains MSG-Min may be more robust, while on the antmaze-large domain, MSG-LCB may lead to better results.

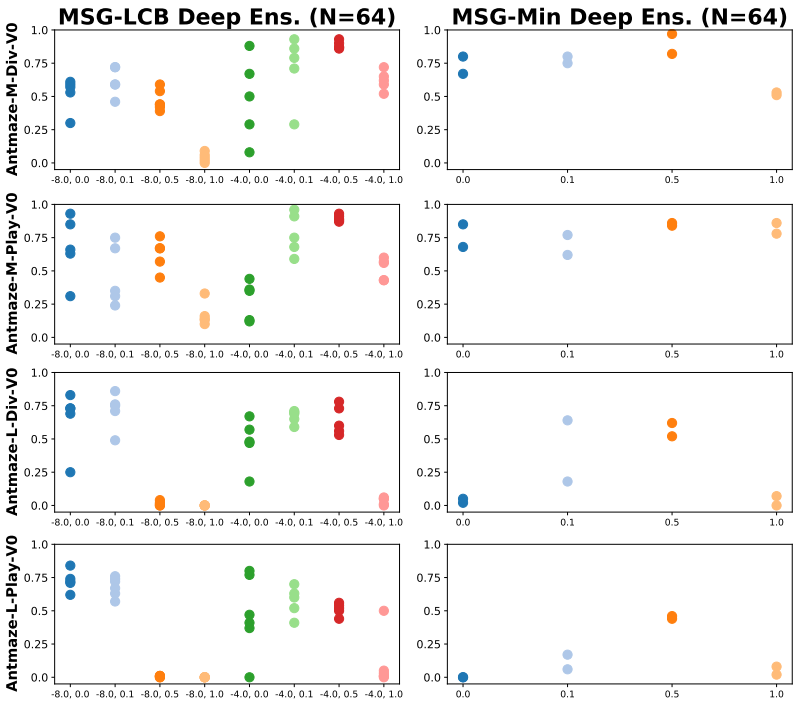


Figure 20: Comparison of LCB and Min formulation of MSG on the D4RL antmaze domains. The hyperparameters for MSG-LCB, and MSG-Min are (β, α) and α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

L Definition of Deep Ensembles

In the deep learning literature, *deep ensembles* refers to the setting where an ensemble of neural networks with identical network architectures are trained using the same data and objective functions, with the only difference in ensemble members being the random weight initialization of the networks [43, 17].

Figure 5: Basline comparison on D4RL Gym domains. The hyperparameters for MSG, CQL, and F-BRC are (β, α), α, and λ respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

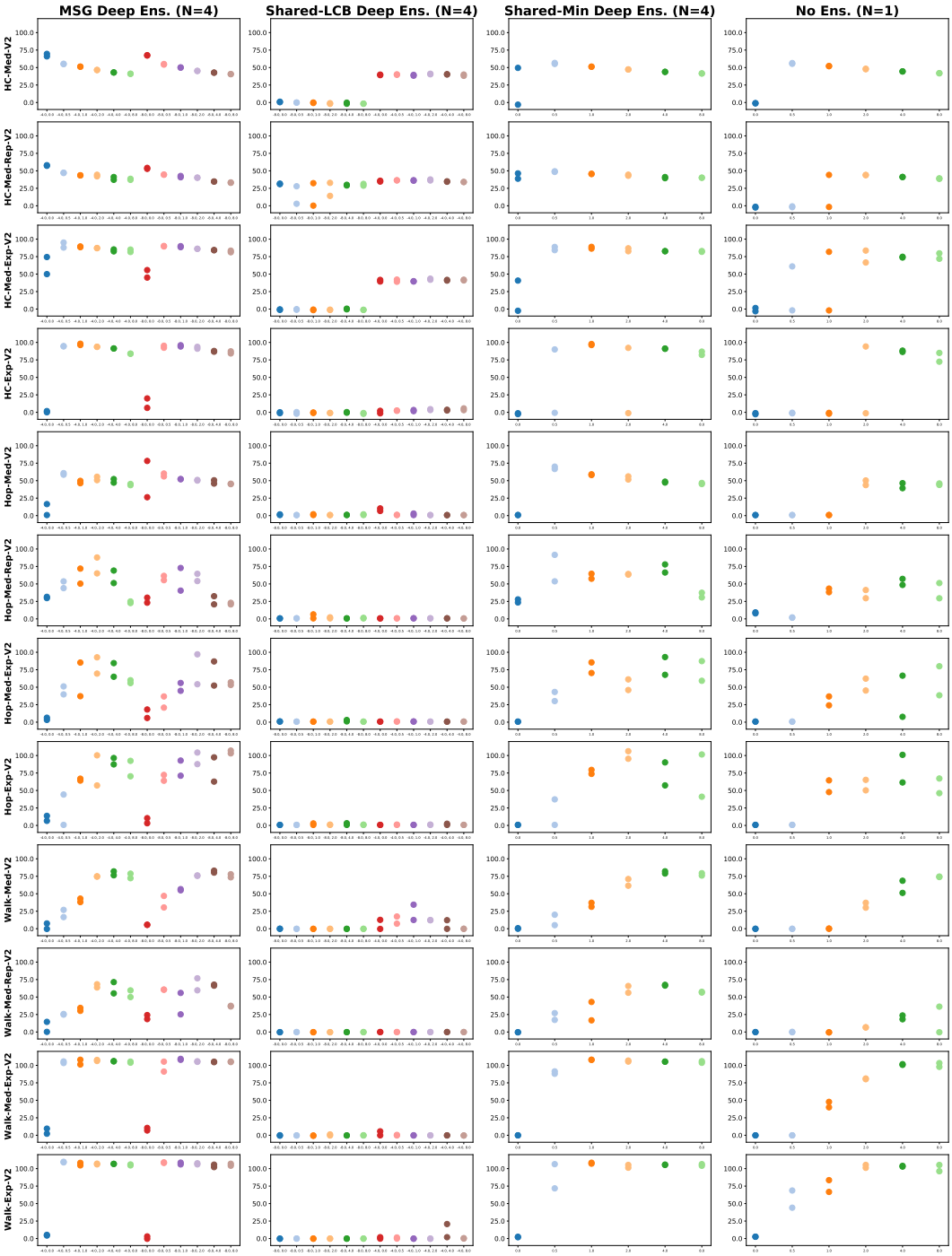


Figure 6: Ablations on D4RL Gym domains. The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

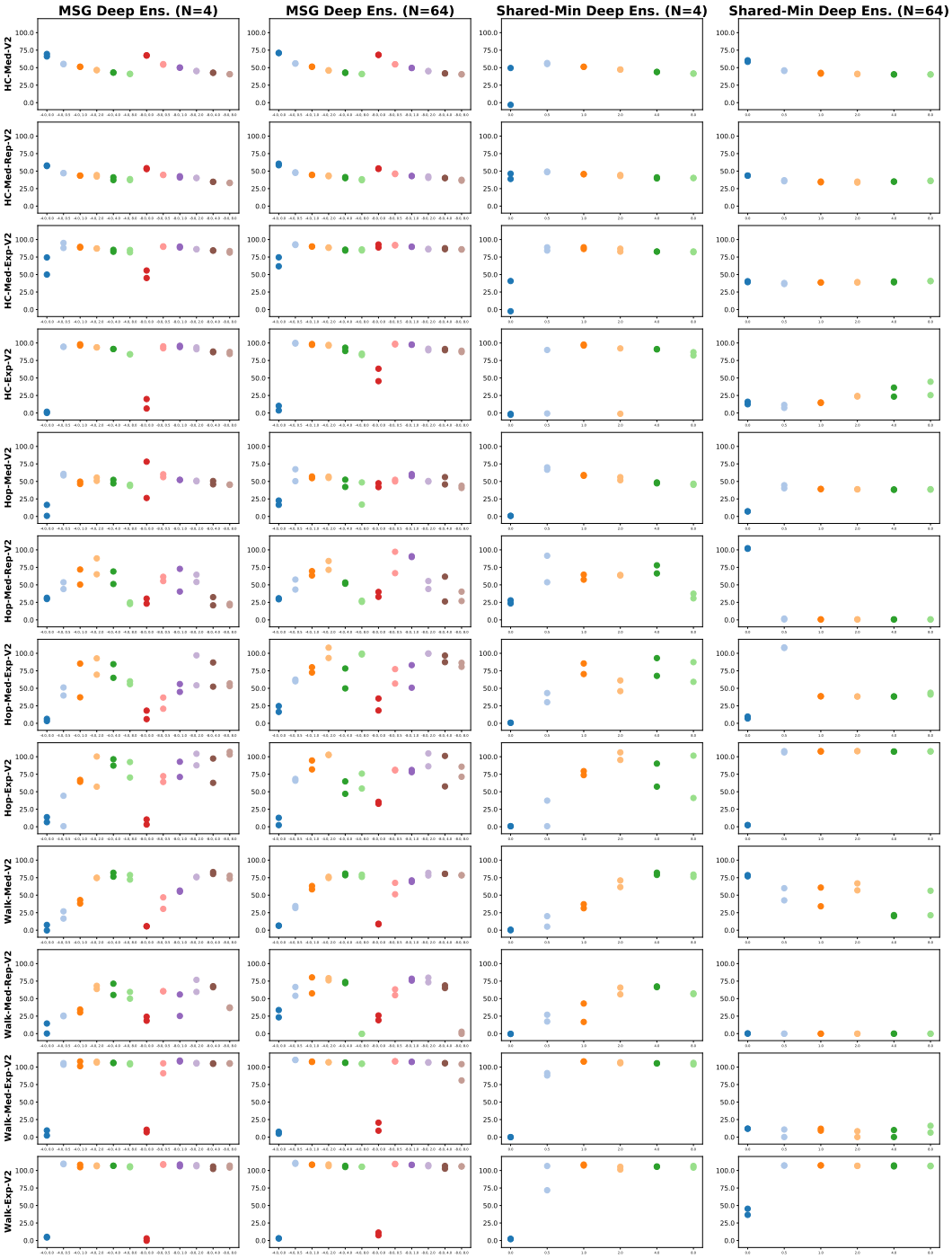


Figure 7: Comparison of MSG and Shared-Min on D4RL Gym domains, for two ensemble sizes of $N = 4$ and $N = 64$. The hyperparameters for MSG and Shared-Min are (β, α) and α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

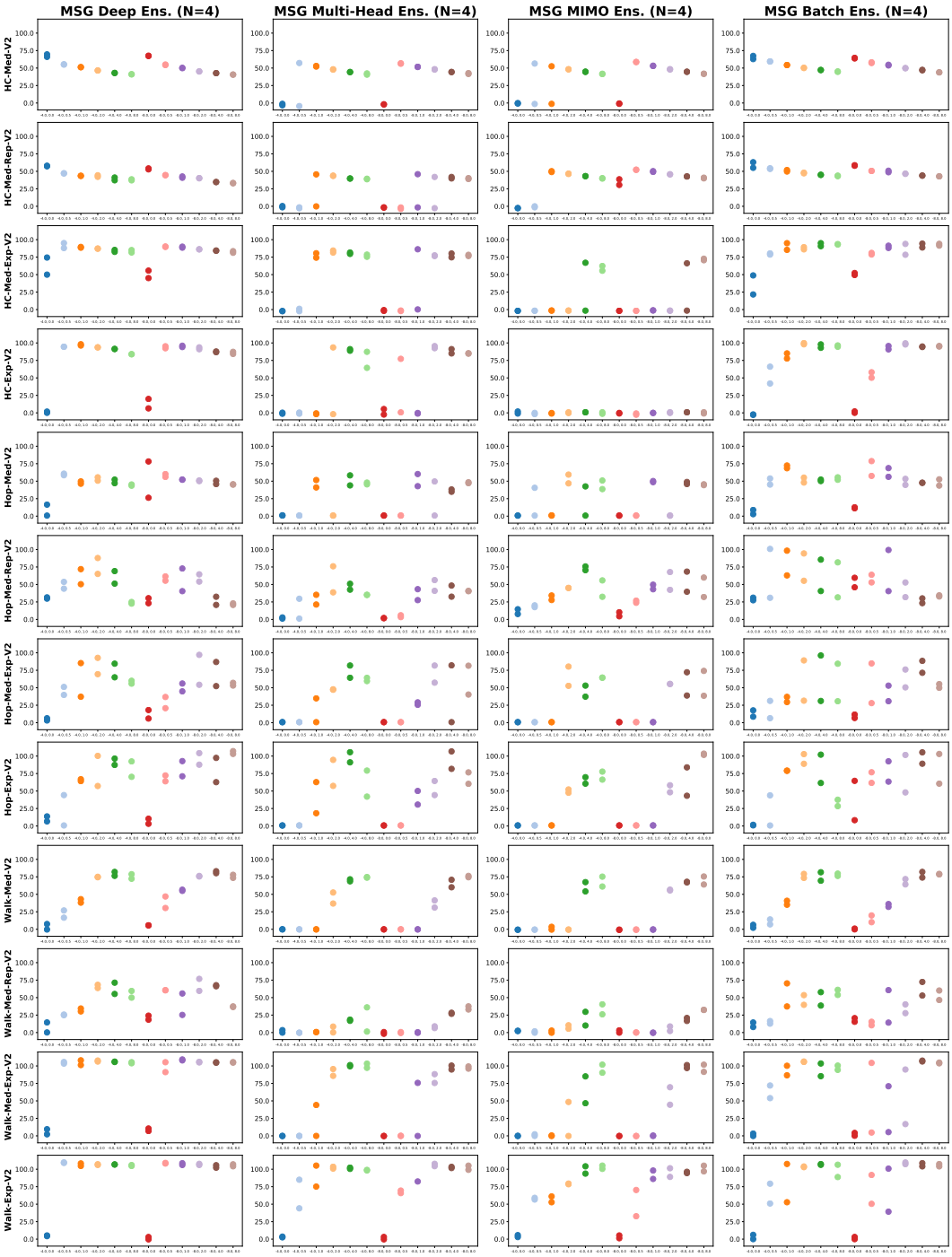


Figure 8: Efficient ensembles on D4RL Gym domains. The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

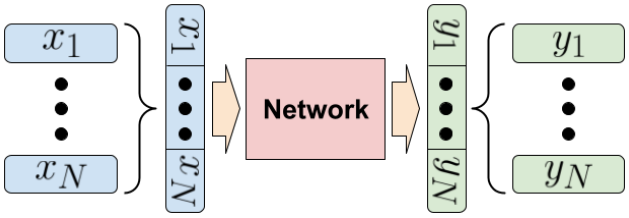


Figure 16: Visual depiction of MIMO Ensemble