A D4RL Gym Locomotion Benchmarks

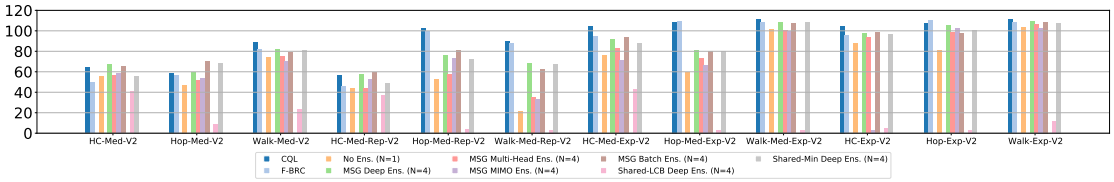


Figure 4: Summary of D4RL Gym benchmark results (full results presented in Appendix A). For each method, we report the mean across random seeds for the best hyperparameter. Numerical results for all experiments are available in the supplementary material.

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

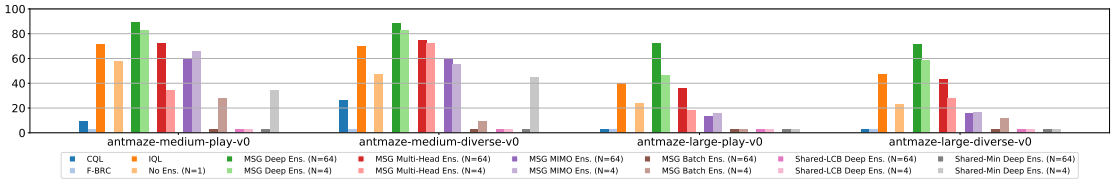


Figure 9: This Figure is identical to Figure 3. Summary of D4RL antmaze benchmark results (full results presented in Appendix B). For each method, we report the mean across random seeds for the best hyperparameter. Numerical results for all experiments are available in the supplementary material.

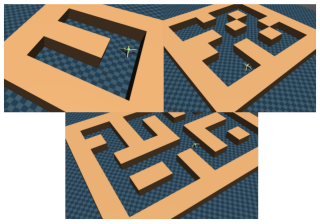


Figure 10: D4RL antmaze tasks. Figure taken from [24].

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.

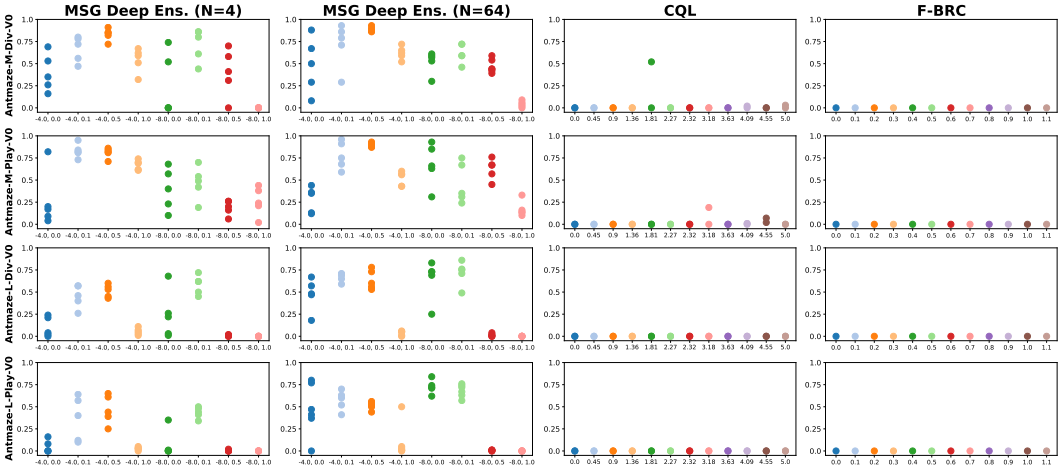


Figure 11: Basline comparison on D4RL antmaze domains. The hyperparameters for MSG, CQL, and F-BRC are (β, α), α, and λ respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

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.

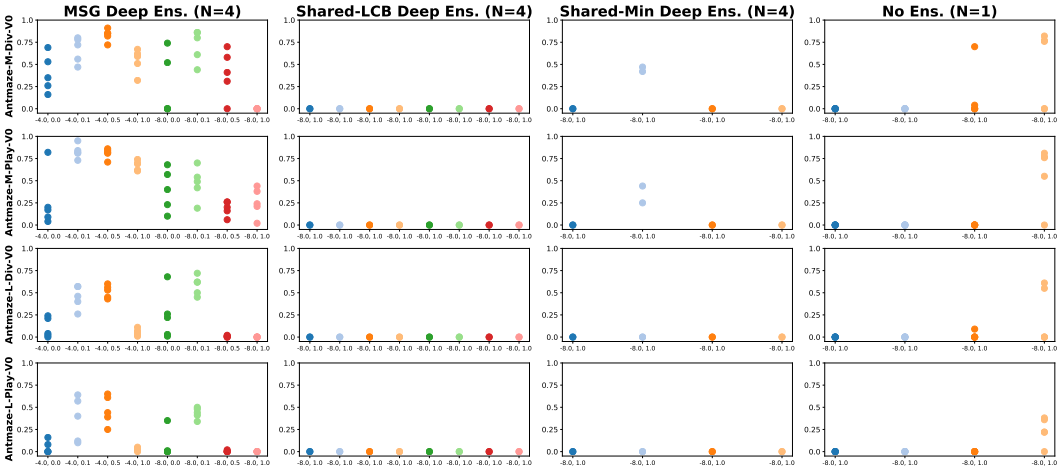


Figure 12: Ablations on D4RL antmaze domains ($N = 4$). The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

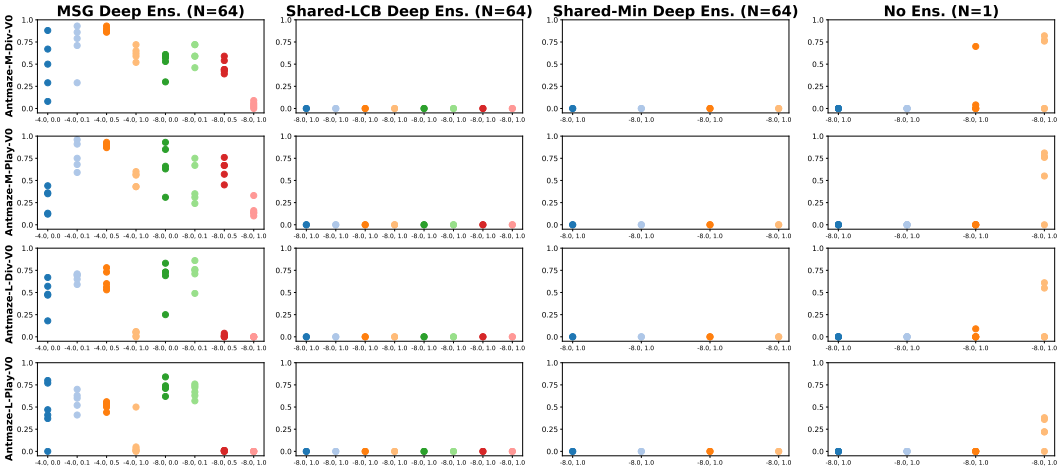


Figure 13: Ablations on D4RL antmaze domains ($N = 64$). The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

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 $N = 64$.

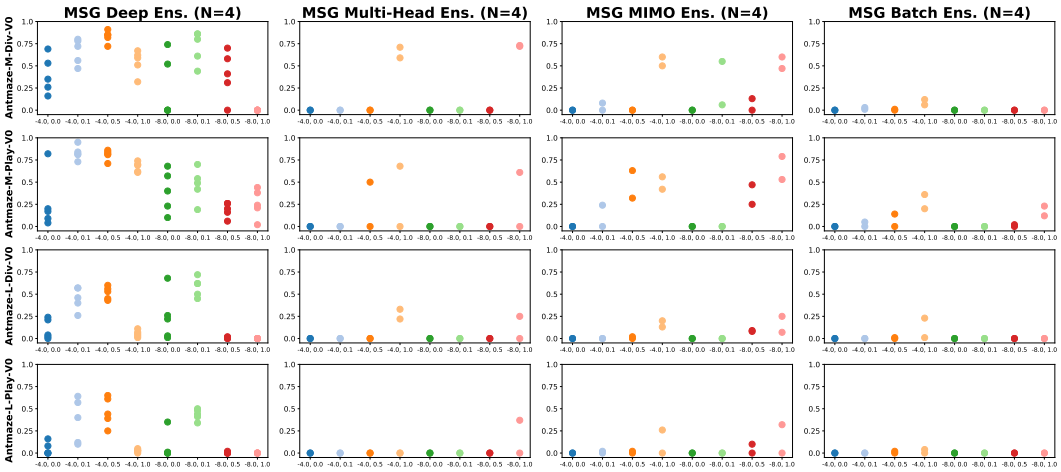


Figure 14: Efficient ensembles on D4RL antmaze domains ($N = 4$). The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

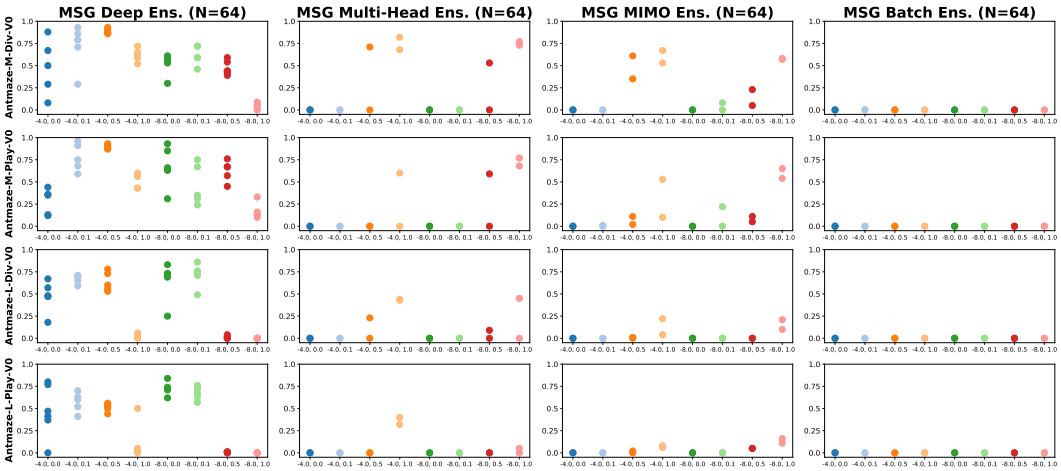


Figure 15: Efficient ensembles on D4RL antmaze domains ($N = 64$). The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

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 */`
  - $\forall s'_m \in batch, a'_{\pi,m} \sim \pi(s'_m)$
  - $L(\theta_i) = \frac{1}{M} \sum_m \left(Q_{\theta^i}(s_m, a_m) - (r + \gamma \cdot Q_{\bar{\theta^i}}(s'_m, a'_{\pi,m})) \right)^2$
  - `/* 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 */`
  - $\mathcal{L}(\theta^i) = L(\theta^i) + \alpha \cdot \mathcal{H}(\theta^i)$
  - $\theta^i \leftarrow \theta^i - \texttt{AdamUpdate}\left(\mathcal{L}(\theta^i), \theta^i\right)$
  - $\bar{\theta^i} \leftarrow \tau \cdot \bar{\theta^i} + (1 - \tau) \cdot \theta_i$

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

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

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,

$$Q^{(t+1)}_{\text{lin}}(\mathcal{X}') = Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}', \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(\mathcal{Y}^{(t)} - Q^{(0)}(\mathcal{X})\right) \tag{13}$$

$$= Q^{(0)}(\mathcal{X}') + \hat{\Theta}^{(0)}\left(\mathcal{X}', \mathcal{X}\right) \cdot \hat{\Theta}^{(0)}\left(\mathcal{X}, \mathcal{X}\right)^{-1} \cdot \left(R + \gamma Q^{(t)}_{\text{lin}}(\mathcal{X}') - Q^{(0)}(\mathcal{X})\right) \tag{14}$$

$$= Q^{(0)}(\mathcal{X}') + C \cdot \left(R + \gamma Q^{(t)}_{\text{lin}}(\mathcal{X}') - Q^{(0)}(\mathcal{X})\right) \tag{15}$$

$$= Q^{(0)}(\mathcal{X}') + CR - CQ^{(0)}(\mathcal{X}) + \gamma C Q^{(t)}_{\text{lin}}(\mathcal{X}') \tag{16}$$

$$= \dots \tag{17}$$

$$= (1 + \ldots + \gamma^t C^t)\left(Q^{(0)}(\mathcal{X}') + CR - CQ^{(0)}(\mathcal{X})\right) + (\gamma C)^{t+1} Q^{(0)}(\mathcal{X}') \tag{18}$$

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,

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

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,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$= \ldots \tag{36}$$

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

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

Based on this form, we can write,

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

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

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

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

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

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

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

Thus, we have our results,

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

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

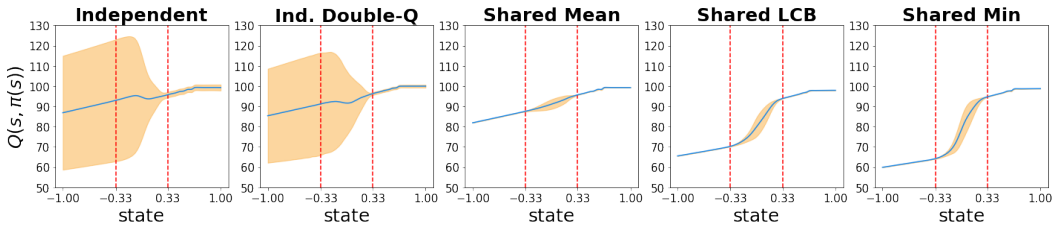


Figure 17: Verifying theoretical predictions on the toy Continuous Chain MDP. The marked interval $[-0.33, 0.33]$ denotes the region of state-space with no data. As anticipated by Theorem 3.1, when the Q-value functions are trained independently, the derived uncertainties capture the interaction between 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 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:** $y^i = r + \gamma \cdot \min\left[Q^1_{\theta^i}(s', \pi(s')), Q^2_{\theta^i}(s', \pi(s'))\right]$
- **Shared Mean Targets:** $y = r + \gamma \cdot \text{mean}\left[Q_{\theta^i}(s', \pi(s'))\right]$
- **Shared LCB Targets:** $y = r + \gamma \cdot \left[\text{mean}\left[Q_{\theta^i}(s', \pi(s'))\right] - 2 \cdot \text{std}\left[Q_{\theta^i}(s', \pi(s'))\right]\right]$
- **Shared Min Targets:** $y = r + \gamma \cdot \min\left[Q_{\theta^i}(s', \pi(s'))\right]$

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, $\forall s, \pi(s) = 0.1$.

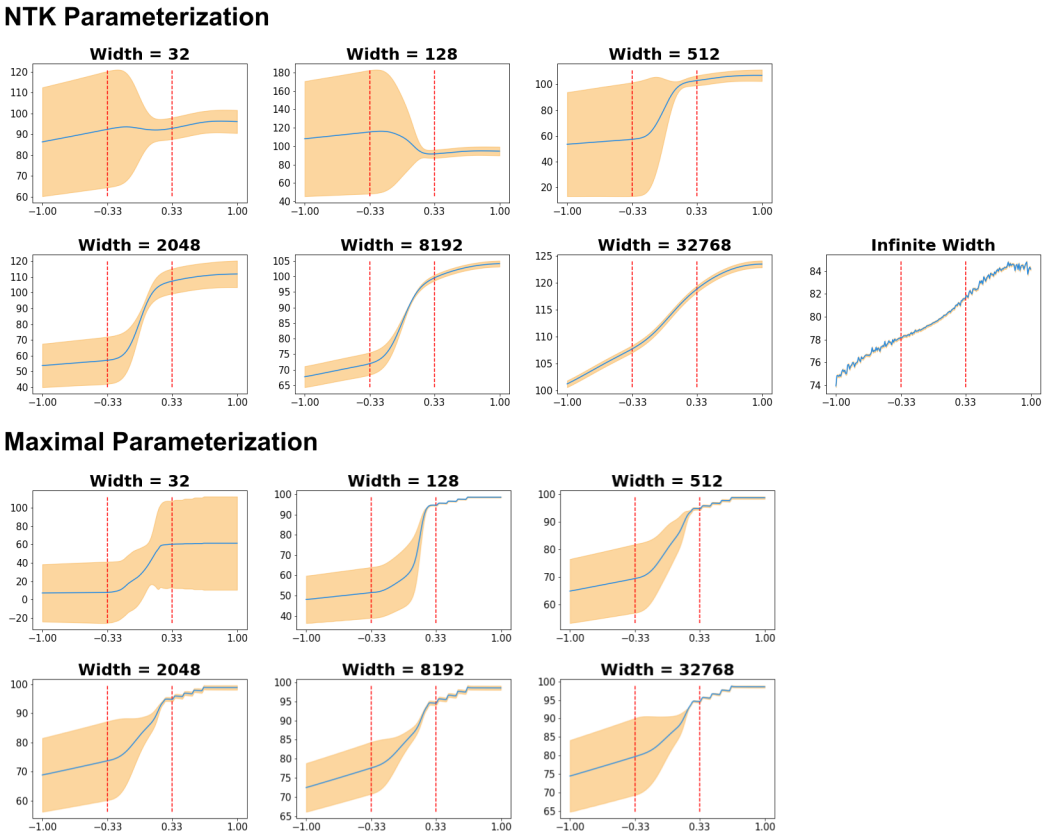


Figure 18: Comparing results of finite-width networks to closed form equations derived in Theorem 3.1. In the NTK parameterization, as width $\to \infty$, the structure of the variances collapse and resemble the infinite-width closed-form results. We believe this is due to infinite-width networks under 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

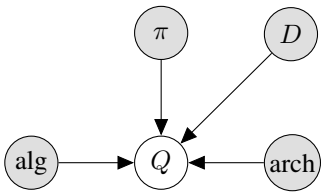
To evaluate the quality of uncertainties obtained from different Q-function ensembling approaches, we create $N = 64$ Q-function networks, each being a one hidden layer neural network with hidden dimension 512 and `tanh` activation. The initial weight distribution is a fan-in truncated normal distribution 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

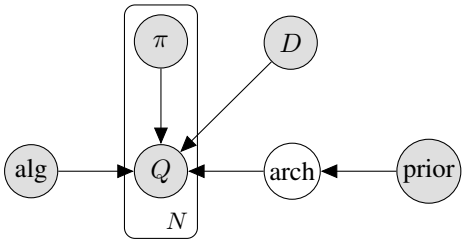
Throughout this work, we have reported the results for every method, domain, hyperparameter choice, and random seed. Based on our experience, in this section we outline a practical workflow for applying MSG to new offline RL environments and datasets.

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.

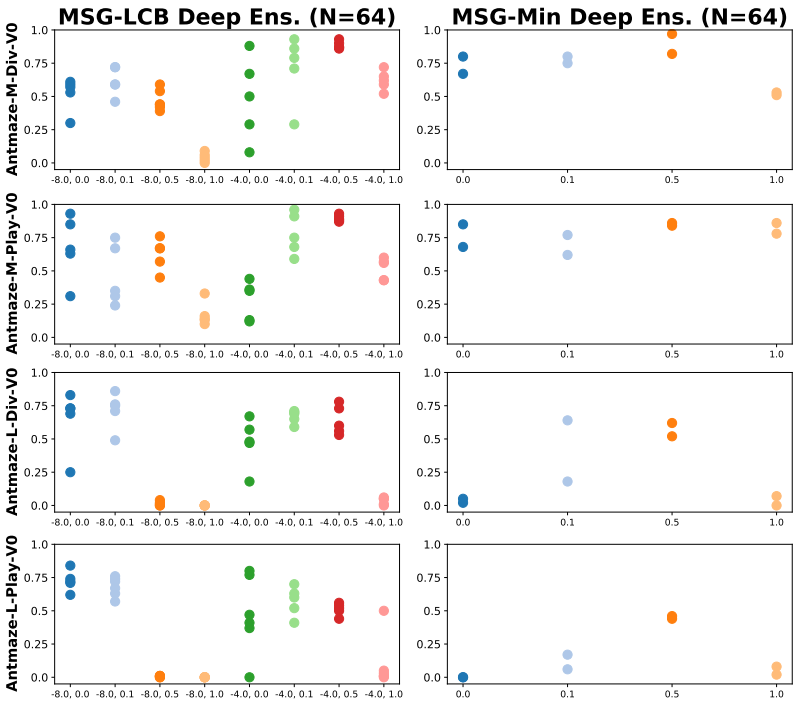


Figure 20: Comparison of LCB and Min formulation of MSG on the D4RL antmaze domains. The hyperparameters for MSG-LCB, and MSG-Min are (β, α) and α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

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.

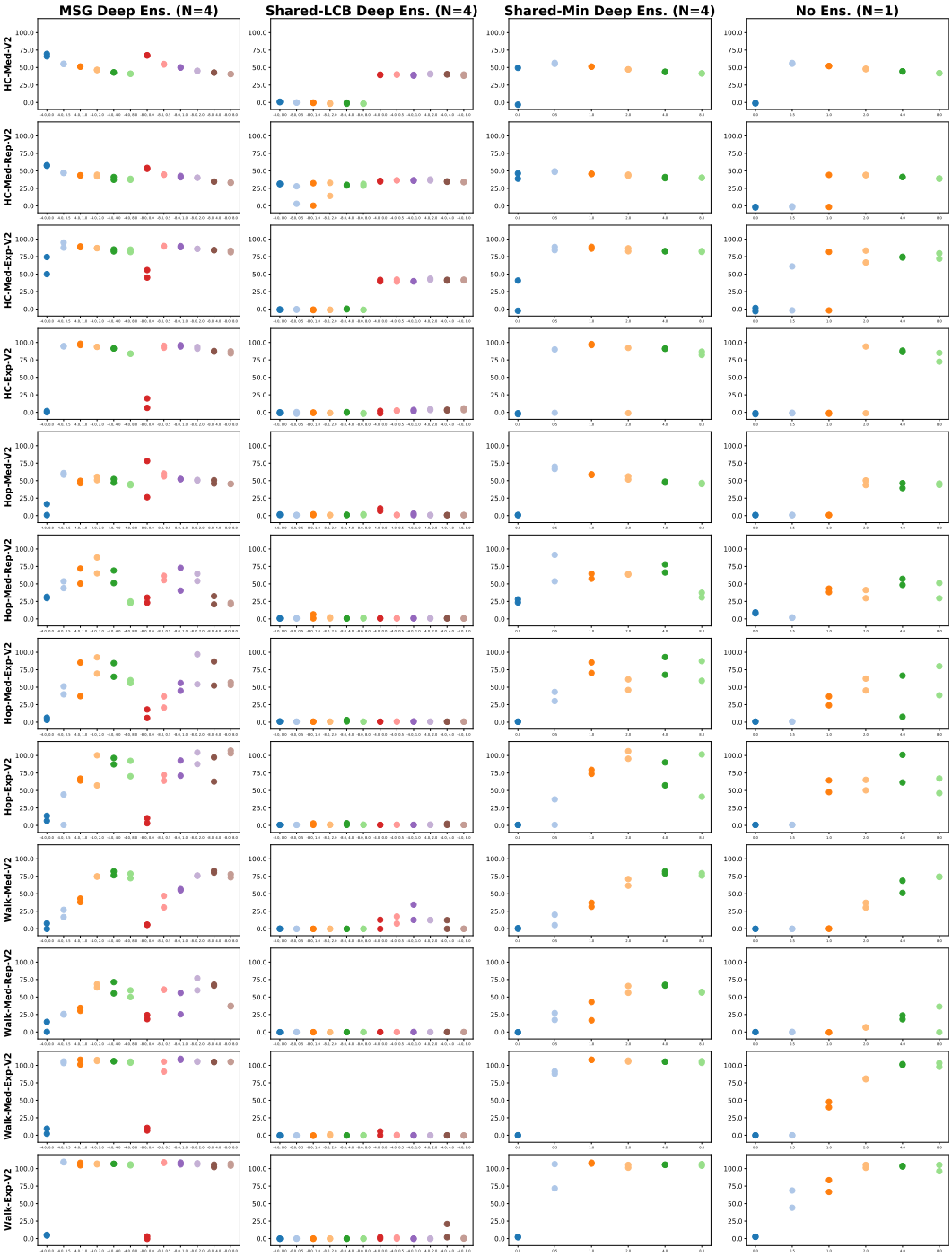


Figure 6: Ablations on D4RL Gym domains. The hyperparameters for MSG, Shared-LCB, Shared-Min, and No Ensemble are (β, α), (β, α), α, α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

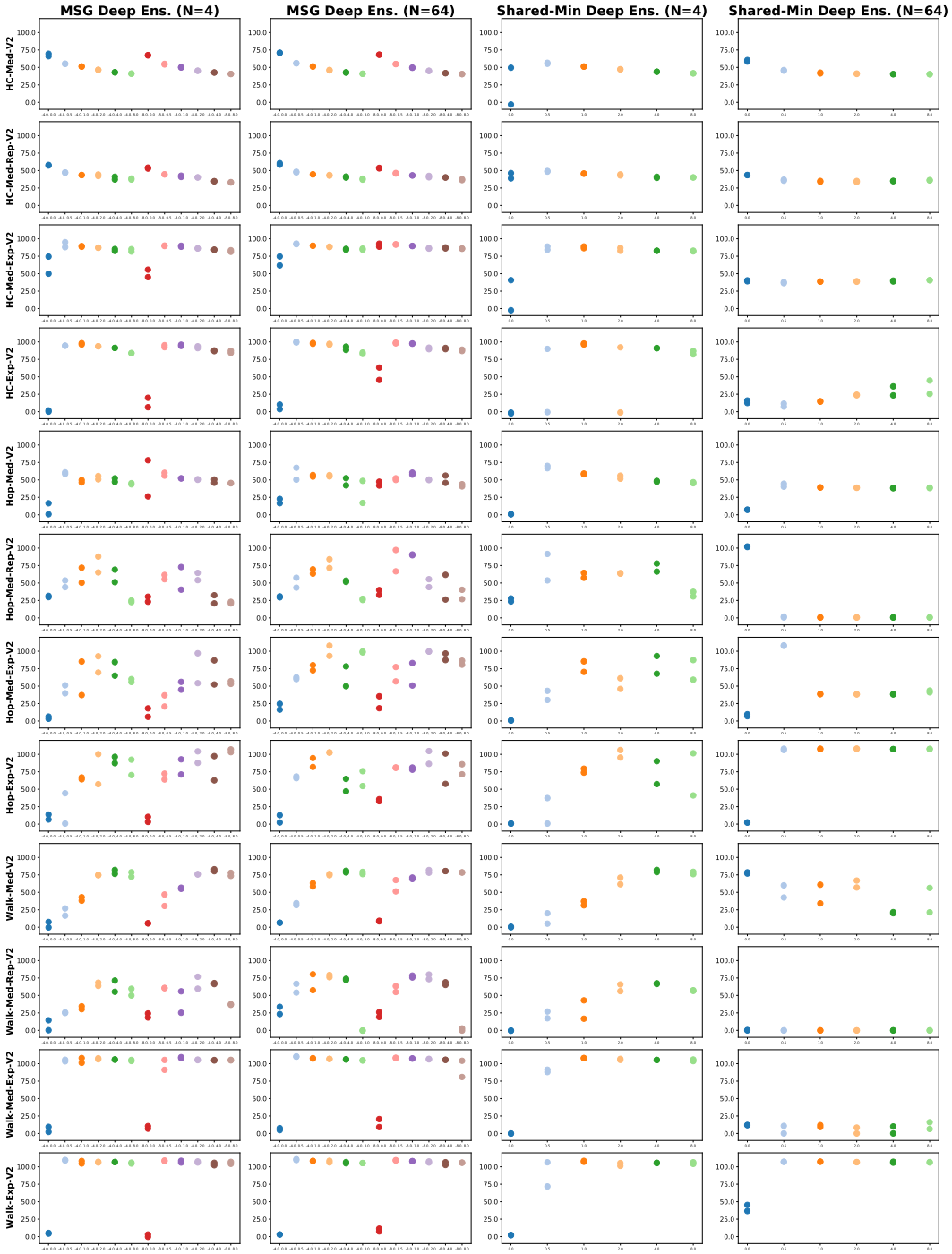


Figure 7: Comparison of MSG and Shared-Min on D4RL Gym domains, for two ensemble sizes of $N = 4$ and $N = 64$. The hyperparameters for MSG and Shared-Min are (β, α) and α respectively. For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

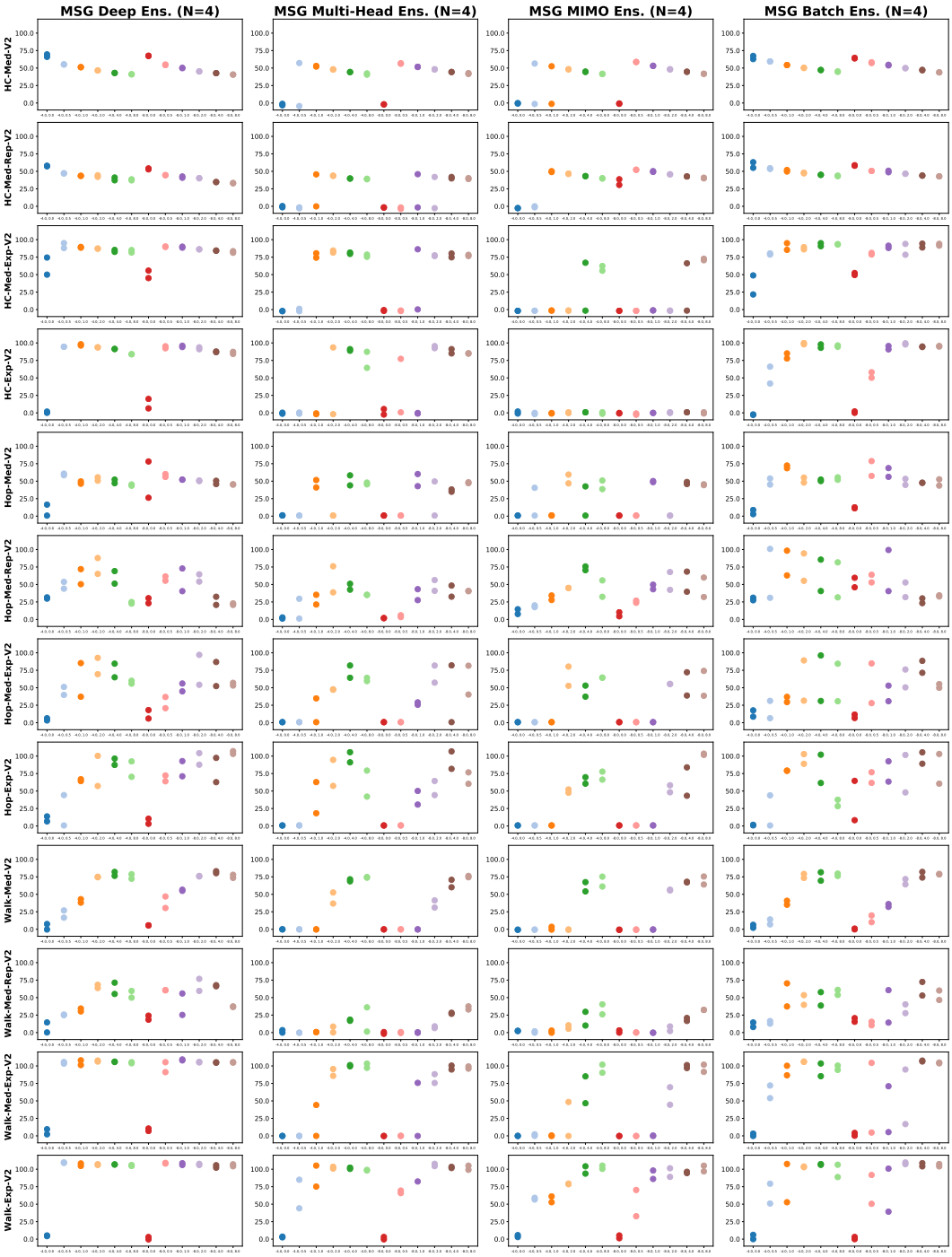


Figure 8: Efficient ensembles on D4RL Gym domains. The hyperparameters for MSG are (β, α). For legibility of hyperparameter values on the x-axis, please zoom into the pdf document.

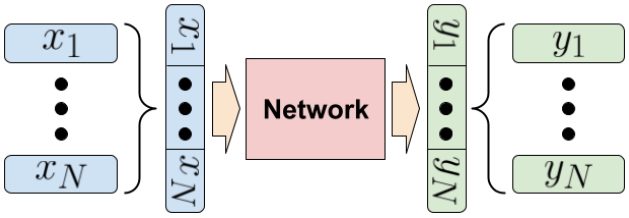


Figure 16: Visual depiction of MIMO Ensemble