Appendix

Table of Contents

A Proofs

A.1 The SoftMax Location of Gumbel Distribution

We refer to the $\log\text{-sum-}\exp$ operator as the SoftMax operator. This is not the same-named operator in Bridle (1989, 1990), which we suggest should be (re-)named SoftArgMax.

Theorem 1.

$$\max_i[\alpha_i + g_i] = \beta \log \sum_i \exp\left(\frac{\alpha_i}{\beta}\right) + g, \quad \textit{where } \; g, g_i \sim \mathcal{G}(0, \beta)$$

where $\mathcal{G}$ is a Gumbel distribution. A Gumbel random variable $g \sim \mathcal{G}(\alpha, \beta)$ specified by location $\alpha \in \mathbb{R}$ and spread $\beta > 0$ has PDF $p(g) = \frac{1}{\beta}\exp(-z - \exp(-z))$ with $z = (g - \alpha)/\beta$ and CDF $P(g) = \frac{1}{\beta}\exp(-\exp(-z))$ (Gumbel, 1935).

Proof. First note

$$\alpha_i + g_i, \;\; g_i \sim \mathcal{G}(0, \beta) \implies \alpha_i + g_i \sim \mathcal{G}_i(\alpha_i, \beta)$$

Then, denoting $y = \max_i[\alpha_i + g_i]$

$$\begin{aligned}
P(X \leqslant y) &= \prod_i P_i(X \leqslant y) \\
&= \prod_i \mathcal{G}_i(X \leqslant y) \\
&= \prod_i \exp\left(-\exp\left(-\frac{x - \alpha_i}{\beta}\right)\right) \\
&= \exp\left(-\sum_i \exp\left(-\frac{x - \alpha_i}{\beta}\right)\right) \\
&= \exp\left(-\exp\left(-\frac{x}{\beta}\right)\sum_i \exp\left(\frac{\alpha_i}{\beta}\right)\right) \\
&= \exp\left(-\exp\left(-\frac{x}{\beta} + \log\sum_i \exp\left(\frac{\alpha_i}{\beta}\right)\right)\right) \\
&= \exp\left(-\exp\left(-\frac{1}{\beta}\left(x - \beta\log\sum_i \exp\left(\frac{\alpha_i}{\beta}\right)\right)\right)\right) \\
&= \mathcal{G}\left(\beta\log\sum_i \exp\left(\frac{\alpha_i}{\beta}\right), \beta\right) \\
&= \beta\log\sum_i \exp\left(\frac{\alpha_i}{\beta}\right) + g, \quad \text{where } \; g \sim \mathcal{G}(0, \beta)
\end{aligned}$$

□

When applied to discrete Q-Learning, we produce

$$\max_{\mathcal{A}}\left[Q_\theta(s,a) + g_{\theta,a}(s)\right] = \beta_\theta(s)\log\int_{\mathcal{A}_s} \exp\left(\frac{Q_\theta(s,a)}{\beta_\theta(s)}\right) \mathrm{d}a + g_\theta(s) \tag{10}$$

where $g_{\theta,a}(\cdot), g_\theta(\cdot) \sim \mathcal{G}(0, \beta_\theta(\cdot))$, for all $a \in \mathcal{A}$.

We assume that the same result holds for the continuous case if $|\mathcal{A}| < \infty$, an assumption first used in Lemma 1, Appendix B.1, Page 11 of Haarnoja et al. (2018a) to ensure boundedness. Here, we require the output of the $\max$-operator to be bounded, which cannot be the case when the number of its arguments is ∞.

A.2 Shifting the Value Function

Theorem 2.

$$\beta \log \sum_i \exp\left(\frac{\alpha_i + g}{\beta}\right) = \beta \log \sum_i \exp\left(\frac{\alpha_i}{\beta}\right) + g$$

Proof.

$$\begin{aligned}
\beta \log \sum_i \exp\left(\frac{\alpha_i + g}{\beta}\right) &= \beta \log \sum_i \exp\left(\frac{\alpha_i}{\beta} + \frac{g}{\beta}\right) && (11)\\
&= \beta \log \exp\left(\frac{g}{\beta}\right) \sum_i \exp\left(\frac{\alpha_i}{\beta}\right) && (12)\\
&= \beta \log \exp\left(\frac{g}{\beta}\right) + \beta \log \sum_i \exp\left(\frac{\alpha_i}{\beta}\right) && (13)\\
&= \beta \log \sum_i \exp\left(\frac{\alpha_i}{\beta}\right) + g && (14)
\end{aligned}$$

□

A.3 The Difference between Two Gumbel Random Variables is a Logistic

Theorem 3.

$$x_1, x_2 \sim \mathcal{G}(0, \beta) \implies z = x_1 - x_2, \quad z \sim \mathcal{L}(0, \beta)$$

where $\mathcal{L}$ is a logistic distribution. A logistic random variable $l \sim L(\alpha, \beta)$ with location α and spread β has PDF $\frac{\exp(-(l-\alpha)/\beta)}{\beta(1+\exp(-(l-\alpha)/\beta))^2}$ and CDF $\frac{1}{1+\exp(-(l-\alpha)/\beta)}$.

Proof. First, construct the convolution based on the joint PDF

$$\begin{aligned}
\mathcal{L}(z) &= P(Z \leqslant z)\\
&= P(X_1 - X_2 \leqslant z)\\
&= P(X_1 \leqslant z + x_2)\\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{z+x_2} g(x_1)\, g(x_2)\, \mathrm{d}x_1\, \mathrm{d}x_2\\
&= \int_{-\infty}^{\infty} G(z + x_2)\, g(x_2)\, \mathrm{d}x_2 \quad .
\end{aligned}$$

Rewriting x_2 as x yields

$$\begin{aligned}
\mathcal{L}(z) &= \int_{-\infty}^{\infty} G(z + x)\, g(x)\, \mathrm{d}x\\
&= \int_{-\infty}^{\infty} \exp\left(\exp - \frac{z + x}{\beta}\right) \exp\left(-\frac{x}{\beta} - \exp - \frac{x}{\beta}\right) \mathrm{d}x\\
&= \int_{-\infty}^{\infty} \exp\left(-e^{-\frac{x}{\beta}}\left(1 + e^{-\frac{z}{\beta}}\right)\right) e^{-\frac{x}{\beta}}\, \mathrm{d}x\\
&= \int_{0}^{\infty} \frac{1}{\beta} \exp\left(-u\left(1 + e^{-\frac{z}{\beta}}\right)\right) \mathrm{d}u, \ \text{ where } \ u = e^{-\frac{x}{\beta}},\ \mathrm{d}u = -\frac{1}{\beta} e^{-\frac{z}{\beta}}\, \mathrm{d}x\\
&= \frac{1}{\beta} \frac{1}{1 + e^{-\frac{z}{\beta}}} e^{\left(-u\left(1+e^{-\frac{z}{\beta}}\right)\right)} \Bigg|_0^{\infty}\\
&= \frac{1}{1 + e^{-\frac{z}{\beta}}} \quad .
\end{aligned}$$

which is the CDF of $\mathcal{L}(0, \beta)$. □

A.4 Soft Q-Learning Identity

Theorem 4. *For an arbitrary $p(x)$*

$$\beta \log \int \exp\left(\frac{E(x)}{\beta}\right) \mathrm{d}x = \mathop{\mathbb{E}}_{x\sim p(\cdot)}[E(x)] + \beta\,\mathbb{C}[p \,||\, p^\star], \;\; \textit{where} \;\; p^\star(x) = \frac{\exp\frac{E(x)}{\beta}}{\int \exp\frac{E(x)}{\beta}\,\mathrm{d}x}$$

Proof.

$$\begin{aligned}
\mathop{\mathbb{E}}_{p(x)}[E(x)] + \beta\,\mathbb{C}[p \,||\, p^\star] &= \mathop{\mathbb{E}}_{p(x)}[E(x)] - \beta \int p(x) \log \frac{\exp\frac{E(x)}{\beta}}{\int \exp\frac{E(x')}{\beta}\,\mathrm{d}x'}\,\mathrm{d}x \\
&= \beta \int p(x) \log \int \exp \frac{E(x')}{\beta}\,\mathrm{d}x'\,\mathrm{d}x \\
&= \beta \log \int \exp \frac{E(x)}{\beta}\,\mathrm{d}x
\end{aligned}$$

□

When applied to Q-Learning, the following identity produces

$$\beta(s) \log \int \exp\left(\frac{Q_\theta(s,a)}{\beta(s)}\right) \mathrm{d}a = \mathop{\mathbb{E}}_{\pi_\phi(a'|s')}[Q_\theta(s,a)] + \beta(s)\,\mathbb{C}[\pi_\phi \,||\, p_\theta]$$
$$\text{where} \;\; p_\theta(a \mid s) = \frac{\exp\frac{Q_\theta(s,a)}{\beta(s)}}{\int \exp\frac{Q_\theta(s,a')}{\beta(s)}\,\mathrm{d}a'}$$

B Further Theory and Derivations

B.1 Actor Loss

The actor losses used in DoubleGum, SAC, and DDPG are all derived from the same principle. For a given s, the actor loss function should minimizes the following (reverse) KL-Divergence, previously presented in Equation 7.

$$\min_\phi \beta\,\mathbb{D}_{\text{KL}}[\pi_\phi \,||\, p_\theta], \;\; \text{where} \;\; p_\theta(a \mid s) = \frac{\exp Q_\theta^{\text{new}}(s,a)/\beta}{\int \exp Q_\theta^{\text{new}}(s,a')/\beta\,\mathrm{d}a'} \;\; .$$

This simplifies as

$$\begin{aligned}
\min_\phi \beta\;\mathrm{D}_{\text{KL}}[\pi_\phi \,||\, p_\theta] &= \min_\phi \beta \int \pi_\phi(a \mid s) \log \frac{\pi_\phi(a \mid s)}{p_\theta(a \mid s)}\,\mathrm{d}a \\
&= \max_\phi \left[\beta\,\mathbb{H}[\pi_\phi] + \beta \int \pi_\phi(a \mid s) \log \frac{\exp Q_\theta^{\text{new}}(s,a)/\beta}{\int \exp Q_\theta^{\text{new}}(s,a')/\beta\,\mathrm{d}a'}\,\mathrm{d}a\right] \\
&= \max_\phi \left[\beta\,\mathbb{H}[\pi_\phi] + \beta \int \pi_\phi(a \mid s)\,\frac{Q_\theta^{\text{new}}(s,a)}{\beta}\,\mathrm{d}a\right]
\end{aligned}$$

which is then estimated by Monte-Carlo samples from π_ϕ as

$$\max_\phi \mathop{\mathbb{E}}_{\pi_\phi(a|s)}[Q_\theta^{\text{new}}(s,a) - \beta \log \pi_\phi(a \mid s)] \;\; . \tag{15}$$

SAC (Haarnoja et al., 2018a,b) has a policy with learned variance and state-independent β. DDPG (Lillicrap et al., 2015) has a fixed-variance policy which removes the second term in Equation 15 as it is constant with respect to the maximization. DoubleGum has a state-dependent $\beta(s)$, but uses the same actor loss as DDPG because DoubleGum uses a DDPG fixed-variance policy.

B.2 Maximum-Entropy Reinforcement Learning

SACv1 (Haarnoja et al., 2018b) is a special case of DoubleGum and DDPG (Lillicrap et al., 2015) is a special case of SAC. All three continuous control algorithms have an actor and critic loss derived from the same principle. Section B.1 shows this for the actor losses of DoubleGum, SAC, and DDPG. We now relate the critic losses to each other, starting from the most general case, DoubleGum. In continuous control, DoubleGum uses the following noise model, formed from substituting Equation 7 into Equation 5:

$$Q_\theta^{\text{new}}(s,a) + l_{\theta,a}(s) = \mathop{\mathbb{E}}_{p(s'|s,a)}\left[r + \gamma \mathop{\mathbb{E}}_{\pi_\phi(a'|s')}[Q_\theta^{\text{new}}(s',a')] + \gamma\beta_\theta(s)\,\mathbb{C}[\pi_\phi \mid\mid p_\theta]\right] \quad . \tag{16}$$

Here, $l_{\theta,a}(\cdot) \sim \mathcal{L}(0, \beta_\theta(\cdot))$ is a logistic distribution and $p_\theta(a \mid s) \propto \exp \frac{Q_\theta^{\text{new}}(s,a)}{\beta_\theta(s)}$. The DoubleGum critic loss is derived from this noise model by approximating the RHS with Equation 8 and learning θ with moment matching in Section 3.2.

The SAC noise model is derived from Equation 16 in three ways. First, SAC approximates $l_{\theta,a}(s) \sim \mathcal{L}(0, \beta_\theta(\cdot))$ as $n_{\theta,a} \sim \mathcal{N}(0,\sigma)$, motivated by the fact that both distributions have the same mean/mode. Secondly, SAC approximates the DoubleGum state-dependent logistic spread $\beta_\theta(s)$ as temperature parameter β learned not as a part of the critic but by itself with Lagrangian dual gradient descent. Thirdly, SAC breaks down $\mathbb{C}[\pi_\phi \mid\mid p_\theta] = \mathbb{H}[\pi_\phi] + \mathbb{D}_{\text{KL}}[\pi_\phi \mid\mid p_\theta]$ before assuming that the KL-Divergence is negligible, given that it is minimized by the actor loss. These three approximations yield the SAC noise model as

$$Q_\theta^{\text{new}}(s,a) + n_{\theta,a} = \mathop{\mathbb{E}}_{p(s'|s,a)}\left[r + \gamma \mathop{\mathbb{E}}_{\pi_\phi(a'|s')}[Q_\theta^{\text{new}}(s',a') + \beta \log \pi_\phi(a' \mid s')]\right] \quad . \tag{17}$$

MLE of θ wrt the above noise model yields the MSBE critic loss.

DDPG is a special case of SAC that assumes $\beta \to 0$, removing the last term in Equation 17. $\lim_{\beta\to 0} p_\theta(a \mid s)$ becomes deterministic, so π_ϕ may be modelled by a deterministic policy.

B.3 Interpreting the Cross-Entropy as a Pessimism Factor

In continuous control, Fujimoto et al. (2018) introduced Twin Networks, a method that improved sample-efficiency with pessimistic bootstrapped targets computed by returning a sample-wise minimum from an ensemble of two Q-functions. Follow-up work selects a quantile estimate from an ensemble (Kuznetsov et al., 2020; Chen et al., 2021; Ball et al., 2023), which we demonstrate is equivalent to estimating $V_{\theta,\beta}^{\text{soft, new}}$.

Suppose there is an ensemble of n networks where the i^th network follows $Q_{\theta_i}(s,a) = Q_\theta(s,a) + z_i(s,a)$. Here, Q_θ is an 'ideal' function approximator never instantiated nor computed and z is an arbitrary noise source. When n is sufficiently large,

$$\begin{aligned}\min_i \mathop{\mathbb{E}}_{\pi_\phi(a|s)}[Q_{\theta_i}(s,a)] = \min_i \mathop{\mathbb{E}}_{\pi_\phi(a|s)}[Q_\theta(s,a) + z_i(s,a)] &= \mathop{\mathbb{E}}_{\pi_\phi(a|s)}[Q_\theta(s,a)] + \min_i z_i(s) \\ &= \mathop{\mathbb{E}}_{\pi_\phi(a|s)}[Q_\theta(s,a)] - g(s), \quad \text{where } g(s) \sim \mathcal{G}(\alpha(s), \beta(s)) \quad .\end{aligned}$$

A Gumbel random variable $g \sim \mathcal{G}(\alpha, \beta)$ has $\mathbb{E}[g] = \alpha + \gamma_e\beta$, where γ_e is the Euler-Mascheroni constant, so for a deterministic environment the bootstrapped targets become

$$r + \gamma \mathop{\mathbb{E}}_{\pi_\phi(a'|s')}[Q_\theta(s',a')] - \gamma\alpha(s') - \gamma\gamma_e\beta(s') \quad ,$$

recovering Equation 8, the DoubleGum continuous control targets, up to an additive term $\gamma\alpha(s')$, while $-\gamma\gamma_e\beta(s')$ recovers the spread $\gamma c \sigma_\theta(s')$ up to a negative scaling factor, indicating that the default c should be negative. Moskovitz et al. (2021) and Ball et al. (2023) showed that the appropriate ensemble size and selected quantile changes the overestimation bias, so appropriate values would ensure $\alpha(s') = 0$.

B.4 Comparison between DoubleGum and XQL

We present an explanation of Extreme Q-Learning (XQL) as presented in Appendix C.1 of Garg et al. (2023). XQL can be derived from Soft Bellman Equation backups given by

$$Q(s,a) \leftarrow \mathop{\mathbb{E}}_{p(s'|s,a)}[r(s,a,s') + \gamma V^{\text{soft}}(s)], \text{ where } V^{\text{soft}}(s) = \beta \log \sum_{a'} \exp\left(\frac{Q(s',a')}{\beta}\right)$$

and β is a fixed hyperparameter. Computing the log-sum-exp of V^{soft} is intractable in continuous control, as the sum over a' becomes an integral in continuous control tasks.

Garg et al. (2023) present a method of estimating its value using Gumbel regression. Given a (potentially infinite) set of scalars $x \in X$, Gumbel regression provides a method to estimate the numerical value of $\text{log-sum-exp}_\beta(x) = \beta \log \sum_X \exp x/\beta$. Gumbel regression assumes $x \sim \mathcal{G}(\alpha, \beta)$, where $\mathcal{G}$ is a homoscedastic Gumbel distribution, and β is a fixed (hyper)parameter. α estimated by MLE tends towards $\text{log-sum-exp}_\beta(x)$. MLE is performed by numerically maximizing the log-likelihood of a Gumbel distribution, which recovers the LINear-EXponential (LINEX) loss function introduced by Varian (1975).

Garg et al. (2023) incorporate Gumbel regression into deep Q-Learning in two ways, which they name X-SAC and X-TD3. X-SAC combines Gumbel regression to estimate the soft value function used in SACv0 (Haarnoja et al., 2018a). The soft value function $V_\rho^{\text{soft}}(s)$ is a neural network whose parameters ρ are learned by Gumbel regression from $Q_\psi(s,a) \sim \mathcal{G}(V_\rho(s), \beta)$, where ψ are target network parameters. A neural network Q_θ with parameters θ may then be learned by the MSE between itself and $\mathbb{E}_{p(s'|s,a)}[r(s,a,s') + \gamma V_\rho^{\text{soft}}(s)]$. X-SAC is vastly different from DoubleGum, because our algorithm does not estimate the soft value function with a separate neural network.

Gumbel regression is directly used to learn the Q-values in X-TD3. First, the bootstrapped targets are thusly rewritten

$$\begin{aligned} y^{\text{soft}}(s,a) &= \mathop{\mathbb{E}}_{p(s'|s,a)}\left[r + \gamma\beta \log \sum_{a'} \exp\left(\frac{Q_\phi(s',a')}{\beta}\right)\right] \\ &= \mathop{\mathbb{E}}_{p(s'|s,a)}\left[\gamma\beta \log \sum_{a'} \exp\left(\frac{r + \gamma Q_\psi(s',a') - Q_\theta(s,a)}{\gamma\beta}\right)\right] \end{aligned}$$

In environments with deterministic environments, which comprise all environments considered by Garg et al. (2023) and our paper, Lemma C.1 of Garg et al. (2023) provides a method of learning the soft value function with Gumbel regression on $y^{\text{soft}}(s,a) \sim \mathcal{G}(Q_\theta(s,a), \gamma\beta)$. The Gumbel regression objective used in X-TD3 to learn θ is vastly different from the moment matching with the logistic distribution DoubleGum uses to learn θ.

To motivate their use of Gumbel regression, Garg et al. (2023) derived a noise model which they use to present empirical evidence of homoscedastic Gumbel noise. In contrast, we presented empirical evidence of heteroscedastic logistic noise formed from a noise model with two heteroscedastic Gumbel distributions.

C A Discussion on The Convergence of DoubleGum

To the best of our knowledge, there are two types of convergence analysis in Q-Learning: 1) operator-theoretic analysis over tabular Q-functions, and 2) training dynamics of neural network parameters. We believe the second is more appropriate for DoubleGum, because our theory addresses issues in using neural networks (and not tables) for Q-learning. Nevertheless, for completeness, we discuss convergence guarantees for the tabular setting and the function approximation setting. While we can guarantee convergence for the former setting, we have no guarantees for the second.

C.1 Tabular Q-Functions

Appendices B.1 and B.2 present DoubleGum as a MaxEnt RL algorithm. When Q-functions are tabular, Appendix A of Haarnoja et al. (2018a) shows that MaxEnt RL algorithms may be derived

Algorithm 2: DoubleGum Soft Policy Iteration

```
Input: Finite MDP $(\mathcal{S}, \mathcal{A}, r, p)$, initial tables $Q$, $\beta$, $y^{\text{soft}}$, initial policy $\pi$
Output: Optimal Tabular $Q$-function $Q^\star$
1 for training iteration $i$ do
2     for all $s$ do
3         for all $a$ do
4             $y^{\text{soft}}(s,a) \leftarrow \mathbb{E}_{p(s'|s,a)}\left[r(s,a,s') + \gamma\beta_i(s')\log\sum_{a'}\exp\left(\frac{Q_i(s',a')}{\beta_i(s')}\right)\right]$
5             $Q_{i+1}(s,a) \leftarrow y^{\text{soft}}(s,a)$
6         $\beta_{i+1}(s) \leftarrow \frac{\sqrt{3}}{\pi}\sqrt{\mathbb{V}_{a\sim\pi(a|s)}\left[y^{\text{soft}}(s,a)\right]}$
7         define $\pi(a \mid s) \leftarrow \frac{\exp(Q_{i+1}(s,a)/\beta_{i+1}(s))}{\sum_{a'}\exp(Q_{i+1}(s,a')/\beta_{i+1}(s))}$
```

from soft policy iteration. We therefore present a convergence proof for DoubleGum with tabular Q-functions based on soft policy iteration.

DoubleGum treats the return as coming from a logistic distribution and learns its location and spread. In the tabular setting, two tables would need to be learned, $Q(s,a)$ and $\beta(s)$. An algorithm to learn these tables in a finite MDP with soft policy iteration is presented in Algorithm 2. Policy evaluation is done by Lines 4-6 while Line 7 performs policy improvement.

Proof of convergence of Algorithm 2 is similar to the SAC proof of convergence in Appendix B of Haarnoja et al. (2018a). This should not be surprising, given that Appendix B.2 shows SAC as a special case of DoubleGum. We first show that policy evaluation converges and that a new policy found by policy improvement does not reduce the magnitude of the value function.

Lemma 5 (Soft Policy Evaluation)**.** *Consider the Soft Policy Evaluation operator given by*

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}_{p(s'|s,a)}\left[r(s,a,s') + \gamma\beta_i(s')\log\sum_{a'}\exp\left(\frac{Q_i(s',a')}{\beta_i(s')}\right)\right] \quad \textit{over all } (s,a) \textit{ pairs.}$$

$\lim_{i\to\infty} Q_i$ *converges to the soft Q-value.*

Proof. Following Appendix A.4

$$\beta(s)\log\sum_a\exp\left(\frac{Q(s,a)}{\beta(s)}\right) = \mathbb{E}_{\pi(a|s)}[Q_\theta(s,a)] + \beta(s)\,\mathbb{C}[\pi \,||\, p]$$

$$\text{where } \; p(a \mid s) = \frac{\exp(Q(s,a)/\beta(s))}{\sum_{a'}\exp(Q(s,a')/\beta(s))}$$

the bootstrapped targets may be thusly rewritten

$$\begin{aligned}
&\mathbb{E}_{p(s'|s,a)}\left[r(s,a,s') + \gamma\beta(s')\log\sum_{a'}\exp\left(\frac{Q(s',a')}{\beta(s')}\right)\right] \\
&= \mathbb{E}_{p(s'|s,a)}\left[r(s,a,s') + \gamma\mathbb{E}_{\pi(a'|s')}[Q(s',a')] + \beta(s')\,\mathbb{C}[\pi \,||\, p]\right] \\
&= \mathbb{E}_{p(s'|s,a)}\left[r'(s,a,s') + \gamma\mathbb{E}_{\pi(a'|s')}[Q(s',a')]\right]
\end{aligned}$$

where $r'(s,a,s') = r(s,a,s') + \beta(s')\,\mathbb{C}[\pi \,||\, p]$.

Following Lemma 1 in Haarnoja et al. (2018a), Sutton and Barto (1998) gives convergence of $Q_{i+1}(s,a) \leftarrow \mathbb{E}_{p(s'|s,a)}\left[r'(s,a,s') + \gamma\mathbb{E}_{\pi(a'|s')}[Q_i(s',a')]\right]$ □

The proof of Soft Policy Improvement should be identical to SAC, given that Appendix B.1 shows that DoubleGum and SAC use identical actor losses. As such, Lemma 5 can be used in place of Lemma 1 in Theorem 1 of Haarnoja et al. (2018a), thus showing convergence of DoubleGum in the tabular setting.

C.2 Deep Q-Functions

Parameters of the deep Q-function used by DoubleGum in Algorithm 1 are learned by a loss function equivalent to that of heteroscedastic normal regression. Convergence of DoubleGum in the function approximation setting would therefore rely on convergence of heteroscedastic normal regression.

Zhang et al. (2023) introduces PAC-bounds for heteroscedastic normal regression, but on the condition that the mean-estimate is close to the ground truth mean, as mentioned in Paragraph 1 of Section 4. This is empirically achieved by Seitzer et al. (2022), who analyze heteroscedastic normal regression and find that the mean-estimate frequently converges to an underfitting solution. This is because the Negative Log-Likelihood (NLL) of a normal distribution is minimized when the variance becomes large – in Equation 6, this term is denoted with σ_θ^2. As such, changes in Q_θ^{new} will not change the loss function much. To rectify this, Seitzer et al. (2022) multiplies the NLL of the normal with the numerical value of the standard deviation, reducing the dominance of σ_θ on the loss function.

D Further Empirical Evidence for Theoretical Assumptions

D.1 Noise Distributions in Deep Q-Learning

Figure 6 presents graphs corresponding to Figure 1c for all environments considered in this paper. Continuous control results were generated from DoubleGum with default pessimism ($c = -0.1$).

D.2 Adjusting The Pessimism Factor

Figure 7 presents graphs corresponding to Figure 2 for all continuous control environments considered in this paper.

E Further Experimental Details

E.1 Noise Distribution Discrepancy with Extreme Q-Learning

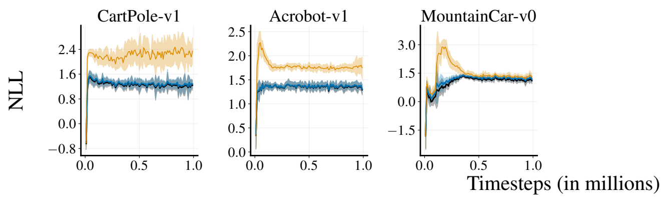
In Appendix D.2 of Page 19, Garg et al. (2023) fitted a Gumbel distribution to the TD errors on three continuous control environments. The Gumbel distribution was a good fit in two of the three environments they benchmarked on. We could not reproduce this result and attribute the discrepancy to experimental differences.

Garg et al. (2023) logged their batch of 256 TD errors once every 5,000 steps during training for 100,000 timesteps, producing ≈ 4000 samples which were aggregated. They also computed bootstrapped targets with online parameters. In contrast, we sample 10,000 TD errors with bootstrapped targets computed from target parameters at a single timestep instance, and we do not aggregate samples across timesteps.

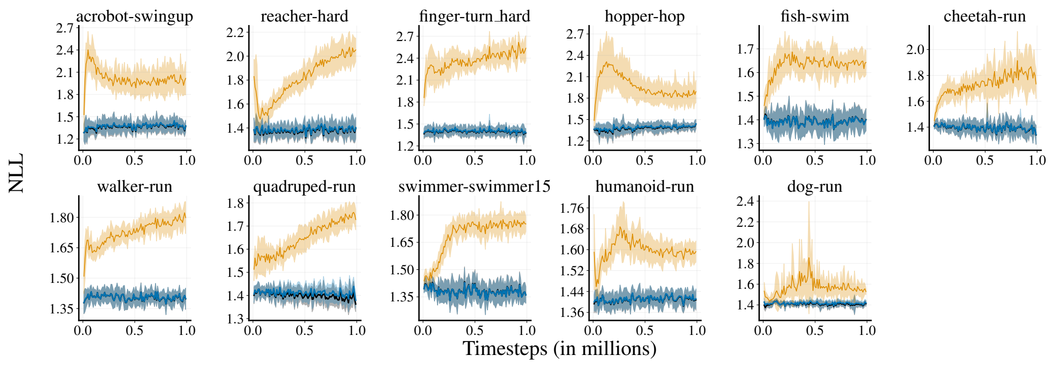
E.2 Continuous Control Benchmarks and Evaluation

As mentioned in Section 6.2, the evaluation metric in continuous control was the normalized IQM with 95% stratified bootstrap confidence intervals from Agarwal et al. (2021). Returns were normalized by a minimum value computed from the mean of 100 rollouts sampled from a uniform policy and the maximum possible return from the environment. When the maximum value was not specified, we used the maximum value of any single rollout attained by any of the baselines.

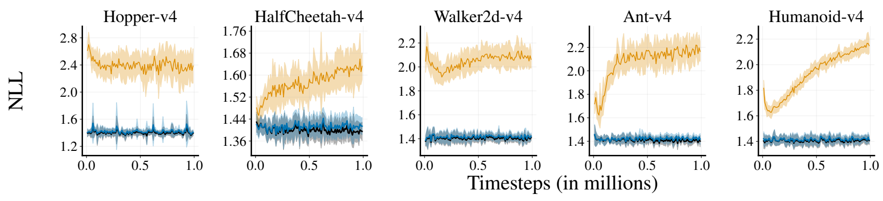
We benchmarked on four continuous control suites: DeepMind Control (Tassa et al., 2018; Tunyasuvunakool et al., 2020), MuJoCo (Todorov et al., 2012; Brockman et al., 2016), MetaWorld (Yu et al., 2020), and Box2D (Brockman et al., 2016). These environments were selected to be as extensive as possible. DeepMind Control and MetaWorld were chosen because of their diversity of tasks, while the MuJoCo and Box2D environments are popular benchmarks within the common interface of OpenAI Gym (Brockman et al., 2016), now Gymnasium (Farama Foundation, 2023). No citation exists for Gymnasium as of writing this paper, and we link to their GitHub repository `https://github.com/Farama-Foundation/Gymnasium` as suggested in `https://github.com/Farama-Foundation/Gymnasium/issues/82`.



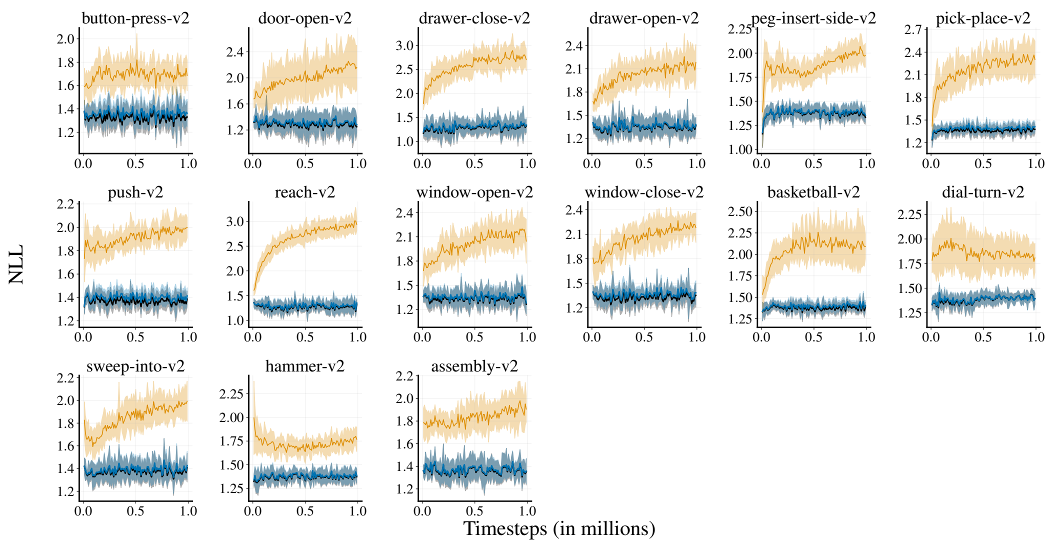
(a) Discrete Control



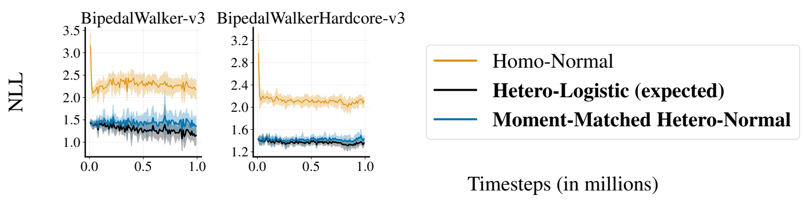
(b) DeepMind Control



(c) MuJoCo

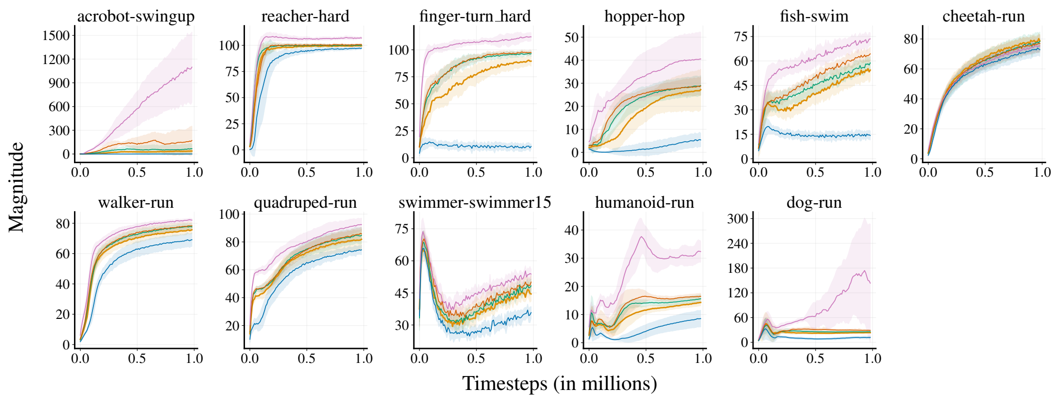


(d) MetaWorld

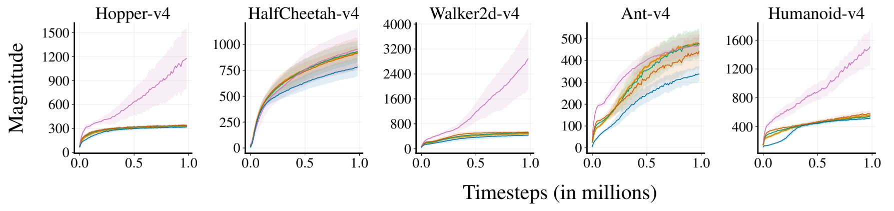


(e) Box2D

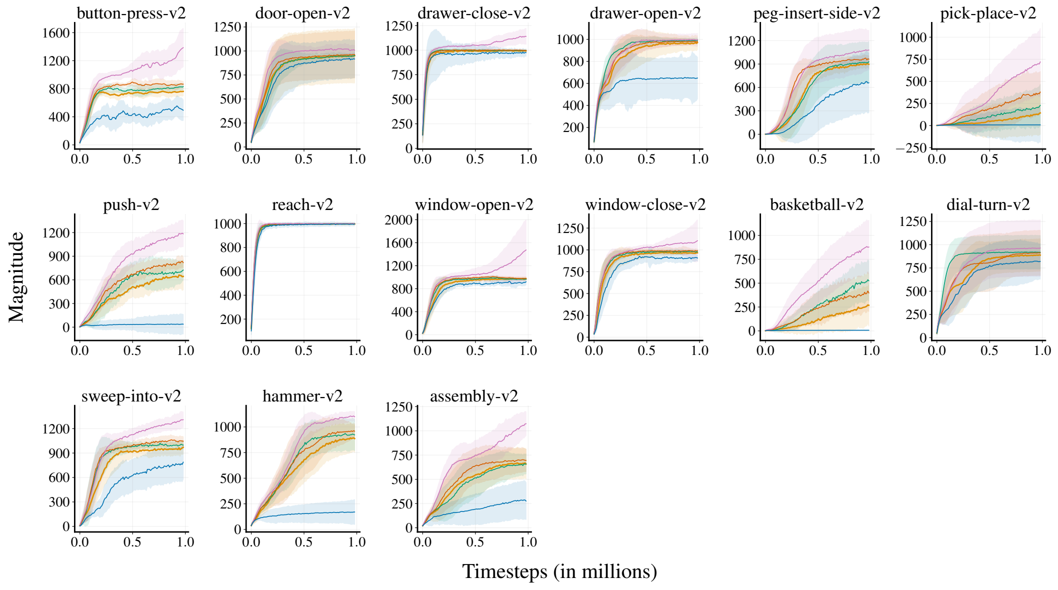
Figure 6: Negative Log-Likelihoods (NLLs) of the noise in Deep Q-Learning under different distributions throughout training (lower is better). Mean calculated per-task $\pm$ standard deviation. The legend for all graphs is in Figure 6e.



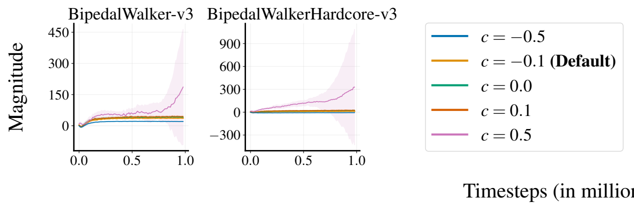
(a) DeepMind Control



(b) MuJoCo



(c) MetaWorld



(d) Box2D

Figure 7: The effect of changing pessimism factor c on the target Q-value in continuous control. IQM calculated per-task $\pm$ standard deviation. The legend for all graphs is in Figure 7d.

DeepMind Control (DMC) was designed to benchmark continuous control, over a broad range of agent morphologies. We selected agent morphologies that could be trained from states with a broad range of action spaces from 1 (`acrobot`) to 38 (`dog`). We did not benchmark on `humanoid_CMU` as this environment was not intended to be solved with RL from scratch, unlike the other baselines. The hardest task was selected from each of the agent morphologies. Properties of the 11 DMC tasks are presented in Table 1a.

MetaWorld was designed to have a diverse range of tasks to evaluate the generalization ability of learned policies. Each environment within MetaWorld is therefore made up of multiple tasks, all with the same underlying structure of an MDP but with different numerical values of their parameters. We follow the method of Seyde et al. (2022) to benchmark on a single MetaWorld task by first selecting an environment and then randomly selecting a set of numerical parameters. Each new instantiation of a MetaWorld task would result in a different set of hyperparameters. As such, we expect the error bars in the aggregate statistics of MetaWorld to be substantially larger than the other environments. We benchmark on tasks formed from the union of the ML1, MT10, and ML10 train tasks that a policy in MetaWorld would be trained on, as well as the five environments benchmarked in Seyde et al. (2022). Properties of the 15 MetaWorld tasks are presented in Table 1c.

MuJoCo was evaluated on the same tasks as SAC (Haarnoja et al., 2018b). These tasks were all locomotion-based. Properties of the 5 MuJoCo tasks are presented in Table 1b.

Box2D was evaluated on all continuous control tasks from states. Properties of the 2 Box2D tasks are presented in 1d.

E.3 Discrete Control Baselines

Discrete control algorithms were implemented as described in Section 3.4. Hyperparameters used in discrete control algorithms are detailed in Tables 2 and 3. We provide explanations for these design choices as follows.

DQN: The original DQN algorithm in Mnih et al. (2015) was designed for pixel inputs. We modified DQN to use state inputs by using an architecture described in Section 3.4 we used in continuous control that was popular for use with state inputs. Conversely to the continuous control architecture, we found removing GroupNorm (Wu and He, 2018) was crucial to getting DQN to work. Similarly to the continuous control architecture, we found that changing the initialization and target network updating drastically improved performance. We also used the MSE and `Adam` (Kingma and Ba, 2014) optimizers as Ceron and Castro (2021) showed that this yielded improved performance over the Huber Loss (Huber, 1992) and RMSProp (Hinton et al., 2012) of the original DQN. Our implementation of DQN solves classic discrete control tasks that the CleanRL (Huang et al., 2022b) reproduction of the original DQN paper at `https://docs.cleanrl.dev/rl-algorithms/dqn/#experiment-results_1` could not solve.

Dueling Double DQN (Dueling DDQN) was a baseline modified from Hessel et al. (2018) designed to be as compatible with DoubleGum as possible. Rainbow evaluated six innovations to DQN: Double DQN (Van Hasselt et al., 2016), Dueling DQN (Wang et al., 2016), noisy networks (Fortunato et al., 2017), n-step returns, C51 distributional RL (Bellemare et al., 2017), and prioritized replay (Schaul et al., 2016). We only used the first two of these six innovations in DoubleGum. We did not find n-step returns effective in discrete domains we considered, nor prioritized replay. Distributional RL was incompatible with DoubleGum, while Schwarzer et al. (2023) did not find noisy networks advantageous.

DoubleDQN was implemented following Van Hasselt et al. (2016) by computing bootstrapped targets of $Q_\psi^{\text{new}}(s, \max_a Q_\theta^{\text{new}}(s,a))$. Dueling DQN was implemented following Wang et al. (2016), with the advantage and value heads having two layers with a hidden layer of size 256 and ReLU activations. The stability of Dueling DQN was greatly improved by setting the biases of both dueling heads to 0.

DoubleGum was implemented as Dueling DDQN with an additional variance head described in Section 3.4.

Table 1: Properties of Continuous Control Environments

(a) DeepMind Control

Environment	Task	Action Dimension	Maximum Return	Minimum Return
`acrobot`	`swingup`	1	1000	3.252
`reacher`	`hard`	2	1000	8.547
`finger-turn`	`hard`	2	1000	67.78
`hopper`	`hop`	4	1000	0.07236
`fish`	`swim`	5	1000	70.99
`cheetah`	`run`	6	1000	3.647
`walker`	`run`	6	1000	22.96
`quadruped`	`run`	12	1000	108.2
`swimmer`	`swimmer15`	14	1000	157
`humanoid`	`run`	21	1000	0.877
`dog`	`run`	38	1000	4.883

(b) MuJoCo

Task	Action Dimension	Maximum Return	Minimum Return
`Hopper-v4`	3	3572	18.52
`HalfCheetah-v4`	6	11960	-283.4
`Walker2d-v4`	6	5737	2.753
`Ant-v4`	8	6683	-60.06
`Humanoid-v4`	17	6829	122.5

(c) MetaWorld

Task	Action Dimension	Maximum Return	Minimum Return
`button-press-v2`	4	10000	187.5
`door-open-v2`	4	10000	277.1
`drawer-close-v2`	4	10000	842.5
`drawer-open-v2`	4	10000	631.8
`peg-insert-side-v2`	4	10000	8.083
`pick-place-v2`	4	10000	5.449
`push-v2`	4	10000	30.62
`reach-v2`	4	10000	776.1
`window-open-v2`	4	10000	230.3
`window-close-v2`	4	10000	306.7
`basketball-v2`	4	10000	10.2
`dial-turn-v2`	4	10000	125.6
`sweep-into-v2`	4	10000	63.41
`hammer-v2`	4	10000	395.1
`assembly-v2`	4	10000	226.3

(d) Box2D

Task	Action Dimension	Maximum Return	Minimum Return
`BipedalWalker-v3`	4	300	-99.97
`BipedalWalkerHardcore-v3`	4	300	-107.9

Table 2: Shared Hyperparameters of Benchmarked Algorithms

Hyperparameter	Value
Evaluation Episodes	10
Evaluation Frequency	Maximum Timesteps / 100
Discount Factor γ	0.99
n-Step Returns	1 step
Replay Ratio	1
Replay Buffer Size	1,000,000
Maximum Timesteps	1,000,000

Table 3: Hyperparameters for Discrete Control

Hyperparameter	Value
Starting Timesteps	2,000
Maximum Timesteps	100,000
Exploration	Policy Churn
Optimizer	`Adam`
Learning rate	3e-4
Number of groups in network GroupNorm	0
Network structure	`Linear(256)`, `ReLU`, `Linear(256)`, `ReLU`

Table 4: Hyperparameters for Continuous Control

Hyperparameter	Value
Starting Timesteps	10,000
Maximum Timesteps	1,000,000
Exploration Noise	0.2
Policy Noise in Critic Loss	0.1
Policy Noise in Actor Loss	0.1
Actor optimizer	`Adam`
Actor learning rate	3e-4
Critic optimizer	`Adam`
Critic learning rate	3e-4
Number of groups in Actor GroupNorm	16
Number of groups in Critic GroupNorm	16
Critic target networks EMA η_ϕ	5e-3
Actor target networks EMA	1
Critic structure	`Linear(256)`, `GroupNorm`, `ReLU` `Linear(256)`, `GroupNorm`, `ReLU`
Actor structure	`Linear(256)`, `GroupNorm`, `ReLU` `Linear(256)`, `GroupNorm`, `ReLU`

Table 5: Pessimism Hyperparameters in Continuous Control

Algorithm	Pessimism Hyperparameter				
	Default	DeepMind Control	MuJoCo	MetaWorld	Box2D
DoubleGum (ours)	-0.1	-0.1	-0.5	0.1	-0.1
DDPG/TD3	Twin	Single	Twin	Single	Twin
SAC	Twin	Single	Twin	Single	Twin
XQL	Twin ($\beta = 5$)	Single (3)	Single (5)	Twin (2)	Twin (5)
QR-DDPG	Single	Single	Twin	Single	Twin
FinerTD3	1	1	3	3	1

E.4 Continuous Control Baselines

Continuous control algorithms were implemented as described in Section 3.4. Hyperparameters used in continuous control algorithms are detailed in Tables 2 and 4. Pessimism hyperparameters are presented in Table 5 and were found following results in Appendix F.2.

As mentioned, all implementations used networks with two hidden layers of width 256, with orthogonal initialization (Saxe et al., 2013) and GroupNorm (Wu and He, 2018). Following Kostrikov (2021), target network parameters were updated with an EMA of $5e-3$ in the critic and 0 in the actor. All these design choices differ from their original implementations but improved aggregate performance. We provide explanations for these design choices as follows.

DDPG was introduced in Lillicrap et al. (2015) and Fujimoto et al. (2018) updated the design choices of DDPG to empirically improve its performance. In addition to the existing changes, our implementation uses the noise clipping scheme in the actor specified by Laskin et al. (2021).

TD3 was implemented with three changes from Fujimoto et al. (2018). First, we update the actor once per critic update – ie using a delay of 1. This is such that the only hyperparameter change between our DDPG and TD3 is the use of Twin Networks. Secondly, we update the actor to maximize the mean of two critics rather than a single critic, a design choice we found empirically reduced variance between training runs. Thirdly, we do not compute the EMA of actor-network parameters. Removing this EMA improves sample efficiency but at the cost of higher variance.

FinerTD3 (our introduced baseline) was implemented with the same hyperparameters as TD3 but with an ensemble of 5 critic networks. We chose to use 5 networks because we tuned the pessimism factor hyperparameter of DoubleGum over 5 values. The 5 critics in FinerTD3 enable five values of pessimism to be used. Pessimism of FinerTD3 is adjusted in the bootstrapped targets. The 5 critic values are sorted by decreasing positivity, and the i^{th} smallest value is used as the target critic value in the bootstrapped targets.

SAC was implemented with hyperparameters from Kostrikov (2021), which we found improved performance. Kostrikov (2021) differs from Haarnoja et al. (2018b) in two additional ways from those mentioned. The standard deviation in the actor was clipped to $[-10, 2]$, and the target entropy was the action dimension divided by 2 instead of just the action dimension.

XQL Garg et al. (2023) presents two off-policy algorithms: X-TD3 and X-SAC. We use X-TD3 to be consistent with the DDPG fixed-variance actor of DoubleGum and refer to it throughout as XQL. XQL tunes two hyperparameters per task: the use of twin networks/not and scalar hyperparameter β. We swept over the same β-values as Garg et al. (2023): 1, 2, 5 without Twin Critics and 3, 4, 10 and 20 with Twin Critics. β was tuned in the same way as pessimism – we found a default β value and a β tuned per-suite. β values are presented in Table 5 and were found following results in Appendix F.2.

MoG-DDPG is formed by combining a Mixture-of-Gaussians (MoG) critic with DDPG. The MoG critic was introduced in Appendix A of Barth-Maron et al. (2018) and improved by Shahriari et al. (2022). The latter paper combines the MoG critic with DDPG with distributed training, but we remove the distributed training component because we do not use it in DoubleGum.

QR-DDPG (our introduced baseline) combines the quantile regression method of Dabney et al. (2018b) with a DDPG actor. Although Ma et al. (2020); Wurman et al. (2022) and Teng et al. (2022) have combined quantile regression with SAC, we combine it with DDPG because DoubleGum is built on top of DDPG. Like Dabney et al. (2018b), we use 201 quantiles, but these are initialized with orthogonal initialization and are optimized with the MSE, rather than the Huber loss. QR was developed for discrete control and uses the Huber loss with the RMSProp optimizer popular in discrete control methods. We found better performance with the MSE and Adam optimizer, perhaps confirming the result of Ceron and Castro (2021) in distributional RL for continuous control.

DoubleGum was implemented as DDPG with a variance head described in Section 3.4.

E.5 Compute Requirements

A single training run for discrete control may take up to 3 to 5 minutes on a laptop with an Intel Core i9 CPU, NVIDIA 1050 GPU and 31.0 GiB of RAM. On the same system, a single training run for continuous control takes 1 - 2 hours.

Table 6: Discrete Control Numerical Results

Task	Score at 100K timesteps (IQM over 12 seeds)		
	DoubleGum (ours)	DQN	DuelingDDQN
`CartPole-v1`	**500** $\pm$ **113.4**	475 $\pm$ 105.5	496.9 $\pm$ 89.1
`Acrobot-v1`	**-62.78** $\pm$ **1.775**	-73.52 $\pm$ 5.191	-64.12 $\pm$ 17.15
`MountainCar-v0`	**-98.17** $\pm$ **2.45**	-99.37 $\pm$ 5.914	-98.75 $\pm$ 30.73

The overwhelming majority of our experiments were run on private infrastructure. This cluster had a mixture of Intel Broadwell, Skylake, Cascade Lake, AMD Rome, AMD Milan CPUs, and NVIDIA P100s, V100s, and A100s GPUs. Benchmarking continuous control took roughly ten times longer than benchmarking discrete control. Multi-threaded experiments for continuous control running twelve seeds in parallel took 5 - 8 hours. 8 algorithms (DoubleGum, DDPG, TD3, MoG-Critics, SAC, XQL, QR-DDPG, FinerTD3) were benchmarked over 33 continuous control environments, and there were further runs for hyperparameter sweeps (4 for DoubleGum, 1 for SAC, 6 for XQL, 1 for QR-DDPG and 4 for FinerTD3), yielding 24 runs in total. These algorithms were run at least 10 times for development and hyperparameter tuning. This yields a lower bound of $8 \times 33 \times 24 \times 10 = 63360$ hours (7.23 years) of computation.

Assuming that all experiments were run on Tesla V100-SXM2-16GB (TDP of 250W), the cluster it was run on had a carbon efficiency of 0.0006 kgCO$_2$eq/kWh (that of the surrounding power grid) and that there were 63360 hours of cumulative computation, the total emissions were 9.51 kgCO$_2$eq, equivalent to driving 36km in an average car. Estimations were conducted using the MachineLearning Impact calculator presented in Lacoste et al. (2019).

F Further Results

F.1 Adjusting the Pessimism of DoubleGum

Figure 8 shows that sample efficiency is sensitive to the pessimism factor c adjusting pessimism per suite greatly impacts sample efficiency. The best performing c was $c = -0.1$, and was thus set as the default pessimism factor value.

Figure 9 shows that the performance of DoubleGum may be improved when the degree of pessimism is changed per suite. This graph was used to determine what pessimism factor to use in each suite, whose values are reported in Table 5.

F.2 Adjusting the Pessimism of Baseline Algorithms

This section presents graphs used to determine which pessimism values to use for baseline algorithms. All final values are reported in Table 5.

Figure 10 shows that sample efficiency is sensitive to the use of pessimism determined by the use of Twin Networks/not. In aggregate, each method was improved by using Twin Networks. Twin networks were therefore set as the default pessimism option for all baseline algorithms apart from QR-DDPG, because Twin Networks was not used with quantile regression in (Dabney et al., 2018b). Figure 11 was used to determine whether to use Twin Networks/not on a per suite basis.

Similarly, Figures 14 and 15 were respectively used to determine pessimism hyperparameters for FinerTD3. In these two graphs, numbers refer to the i^{th} smallest value returned by the ensemble of target critics. Finally, Figures 12 and 13 were respectively used to determine pessimism and β hyperparameters for XQL.

F.3 Discrete Control

Table 6 presents results for discrete control at 100K timesteps.

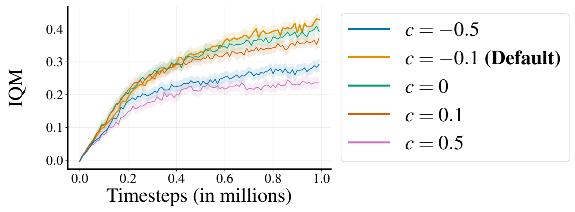


Figure 8: Adjusting the pessimism factor c in DoubleGum, IQM normalized score over 33 tasks in 4 suites with 95% stratified bootstrap CIs.

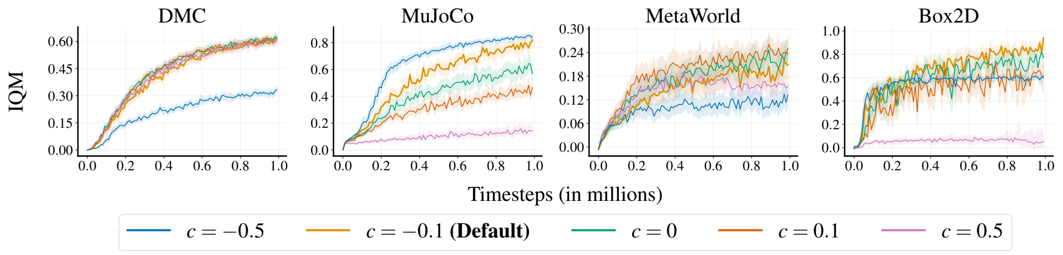


Figure 9: Adjusting pessimism in DoubleGum, per-suite IQM with 95% stratified bootstrap CIs.

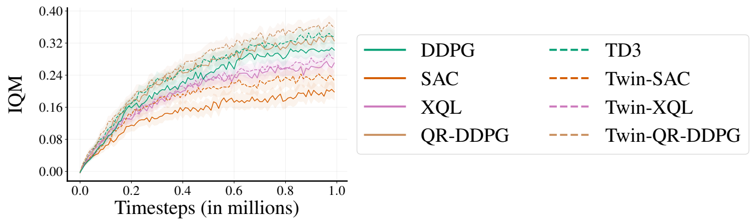


Figure 10: Adjusting pessimism of baseline algorithms with the use of Twin Networks/not, IQM normalized score over 33 tasks in 4 suites with 95% stratified bootstrap CIs. Methods that default to use Twin Networks are dashed.

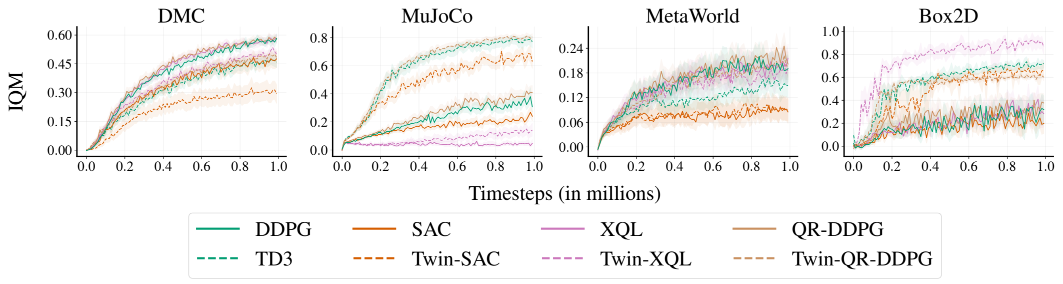


Figure 11: Adjusting pessimism of baseline algorithms with the use of Twin Networks/not, per-suite IQM normalized score with 95% stratified bootstrap CIs. Methods that default to use Twin Networks are dashed.

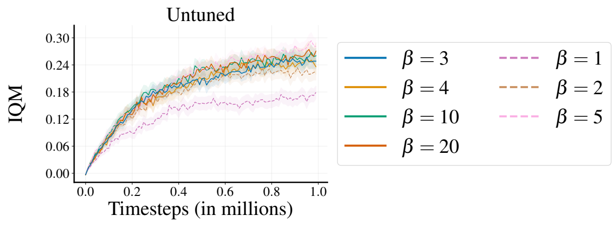


Figure 12: Adjusting pessimism of XQL, IQM normalized score over 33 tasks in 4 suites with 95% stratified bootstrap CIs. Methods that use Twin Networks are dashed.

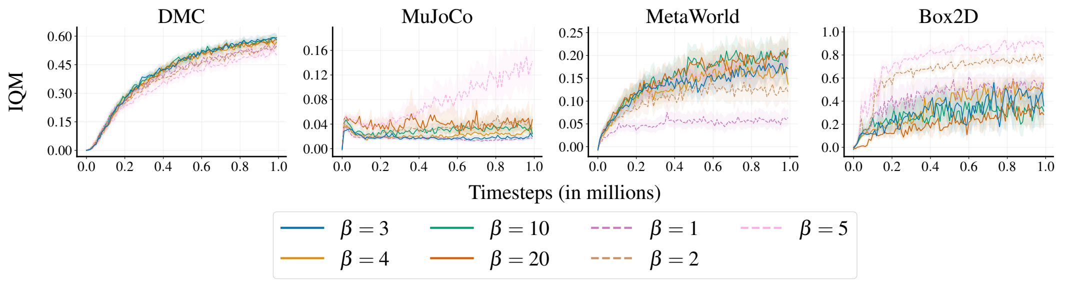


Figure 13: Adjusting pessimism of XQL, per-suite IQM normalized score with 95% stratified bootstrap CIs. Methods that use Twin Networks are dashed.

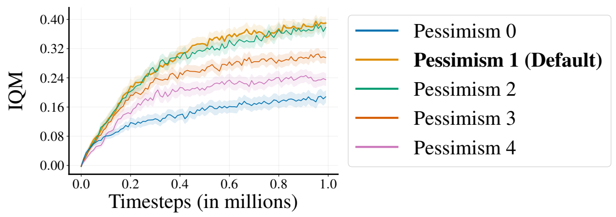


Figure 14: Adjusting pessimism of FinerTD3, IQM normalized score over 33 tasks in 4 suites with 95% stratified bootstrap CIs.

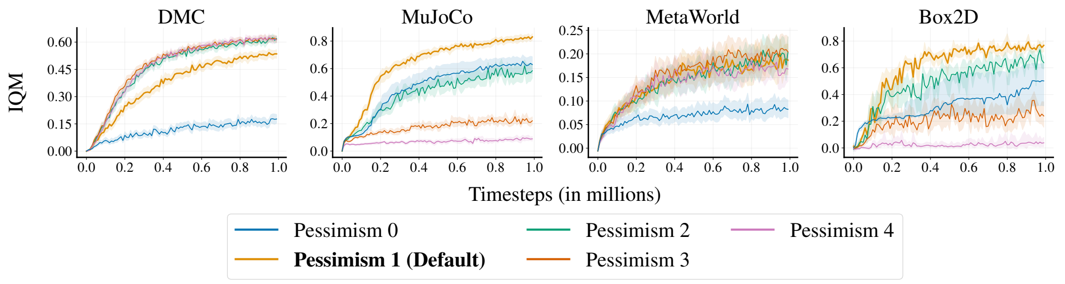


Figure 15: Adjusting pessimism of FinerTD3, per-suite IQM normalized score with 95% stratified bootstrap CIs.

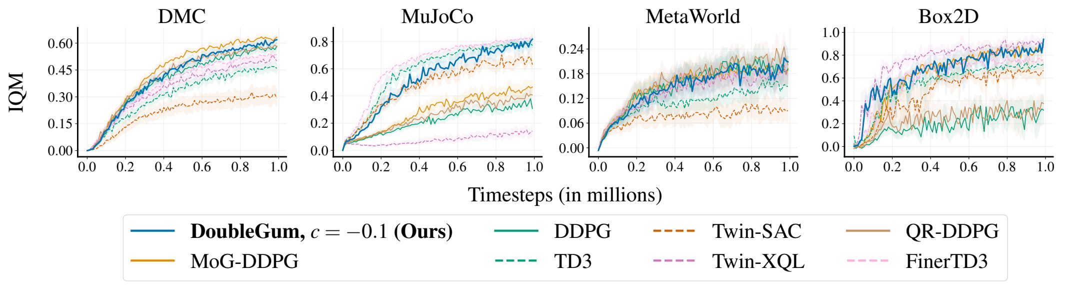


Figure 16: Continuous control with default parameters, per-suite IQM normalized score with 95% stratified bootstrap CIs. Methods that default to use Twin Networks are dashed.

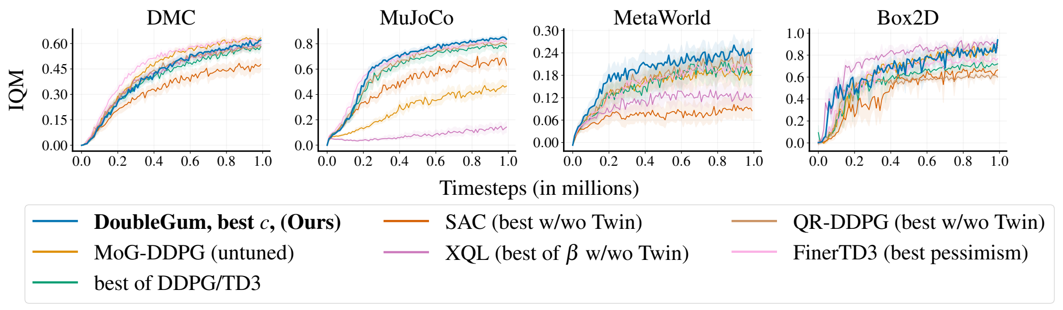


Figure 17: Continuous control with the best pessimism hyperparameters tuned per suite, per-suite IQM normalized score with 95% stratified bootstrap CIs.

F.4 Continuous Control with Default Pessimism

Figures 16 and 18 respectively present aggregate per-suite and per-task results of DoubleGum benchmarked against baseline algorithms with default pessimism values.

Table 7 presents the performance of all algorithms with default pessimism at 1 million timesteps. This graph has six subsections. The first four subsections present per-task results from DeepMind Control, MuJoCo, MetaWorld, and Box2D, respectively, corresponding to results from 18. The next subsection presents per-suite aggregate results, corresponding to Figure 16, while the last subsection presents aggregate results over all tasks and suites, corresponding to Figure 1. In aggregate results, only the IQM is reported.

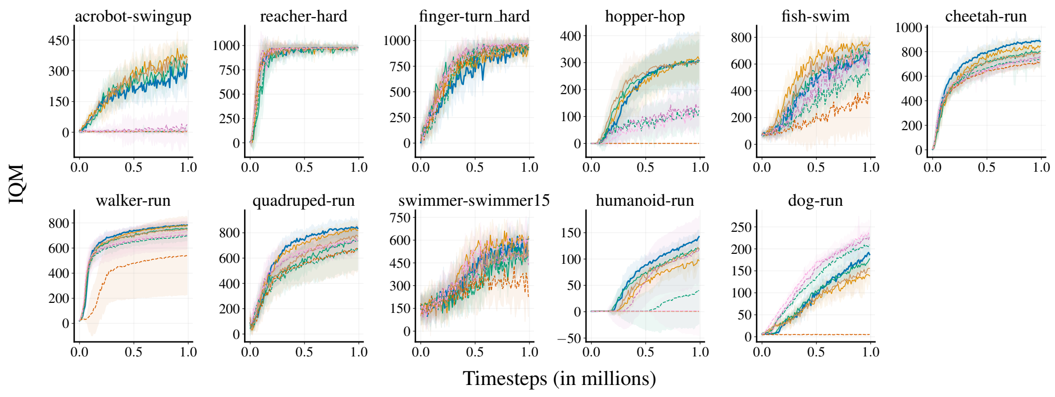
F.5 Continuous Control with Pessimism adjusted Per-Suite

Figures 17 and 19 respectively present aggregate per-suite and per-task results of DoubleGum benchmarked against baseline algorithms with the best pessimism values adjusted per-suite.

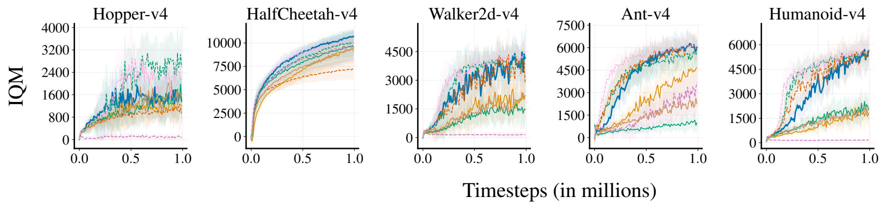
Table 8 presents the performance of all algorithms with default pessimism at 1 million timesteps. This graph has six subsections. The first four subsections present per-task results from DeepMind Control, MuJoCo, MetaWorld, and Box2D, respectively, corresponding to results from 19. The next subsection presents per-suite aggregate results, corresponding to Figure 17, while the last subsection presents aggregate results over all tasks and suites, corresponding to Figure 1. In aggregate results, only the IQM is reported.

Table 7: Continuous control with default pessimism hyperparameters score after 1M timesteps.

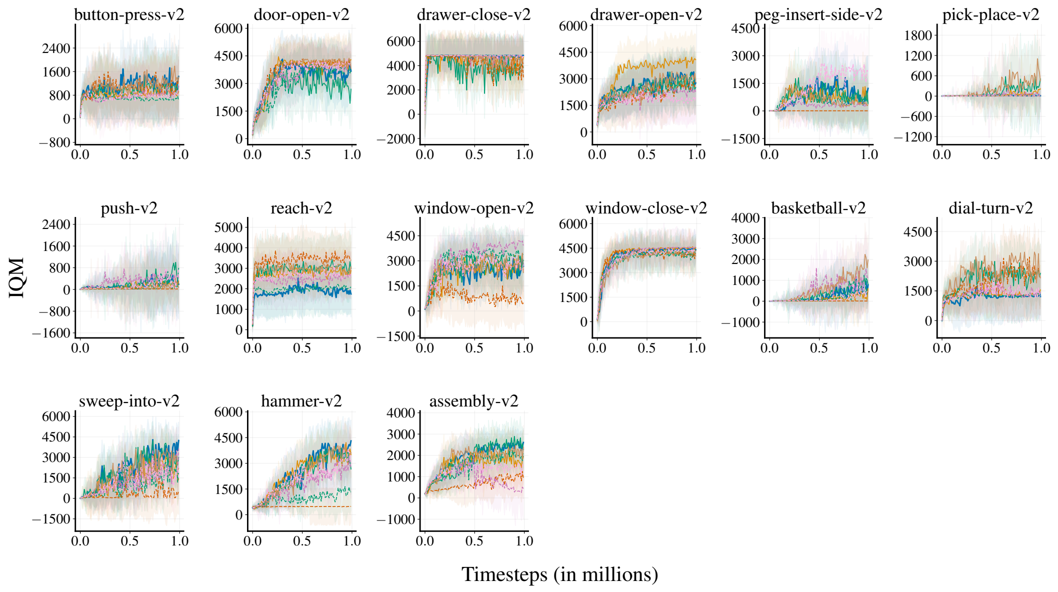
Suite/Task	Score at 1M timesteps (IQM ± standard deviation where appropriate)							
	DoubleGum	MoG-DDPG	DDPG	TD3	Twin-SAC	Twin-XQL	QR-DDPG	FinerTD3
`acrobot-swingup`	330.2 ± 79.44	**364.9 ± 71.75**	334.7 ± 96.92	7.315 ± 26.53	6.568 ± 6.164	38.84 ± 91.41	356.7 ± 88.26	8.114 ± 15.96
`reacher-hard`	**979.4 ± 54.28**	976.9 ± 4.985	975.7 ± 26.78	975.9 ± 31.13	976.2 ± 27.21	975.1 ± 4.506	972.9 ± 14.86	972.4 ± 25.15
`finger-turn_hard`	931.8 ± 98.91	935.4 ± 77.4	909.1 ± 50.68	969.7 ± 28.51	951.5 ± 63.21	967.2 ± 59.48	924.3 ± 73.62	**974.1 ± 24.9**
`hopper-hop`	306.7 ± 101	**313.3 ± 99.74**	305.7 ± 80.12	123.4 ± 69.37	0.007008 ± 0.03587	135.9 ± 65.21	304.3 ± 69.23	116 ± 51.71
`fish-swim`	675.7 ± 58.5	**758.1 ± 62.68**	710.2 ± 94.54	517.9 ± 111.2	344.9 ± 251.5	632.7 ± 99.64	695.2 ± 54.08	675 ± 118.8
`cheetah-run`	**883.1 ± 22.51**	844.9 ± 53.54	804.5 ± 64.31	745 ± 52.78	708.8 ± 44.35	761.3 ± 56.24	785.9 ± 63.43	741.7 ± 43.59
`walker-run`	**783.6 ± 26.97**	778.7 ± 21.86	755.6 ± 28.29	696 ± 39.53	538.6 ± 311.1	705.5 ± 114.6	743.9 ± 29.61	731.4 ± 109.2
`quadruped-run`	**835.1 ± 65.11**	818.2 ± 76.37	736 ± 87.58	677.2 ± 169.6	677 ± 181.7	743.9 ± 100.6	772.4 ± 80.69	739.8 ± 117.1
`swimmer-swimmer15`	608.9 ± 137.8	**623.4 ± 81.98**	531.5 ± 104.1	477.8 ± 139.2	225.4 ± 130.5	435.5 ± 148.5	499.8 ± 64.12	612.5 ± 82.18
`humanoid-run`	**142.5 ± 12.6**	96.97 ± 12.58	119.6 ± 25.09	39.51 ± 62.78	0.8364 ± 0.234	119.2 ± 62.17	117.8 ± 17.68	1.452 ± 47.32
`dog-run`	187.1 ± 13.14	138.8 ± 20.58	177.4 ± 12.17	209.2 ± 25.22	5.226 ± 0.7772	229.5 ± 20.56	155.5 ± 22.55	**234.4 ± 29.06**
`Hopper-v4`	1399 ± 668.1	1211 ± 319.2	1348 ± 740.2	**2589 ± 939.2**	942 ± 135.3	94.89 ± 122.7	1641 ± 554.7	2183 ± 890.6
`HalfCheetah-v4`	**10710 ± 608.3**	9398 ± 1044	9552 ± 1557	10020 ± 1390	7171 ± 1000	9694 ± 1571	9148 ± 1476	10420 ± 328.8
`Walker2d-v4`	4148 ± 1439	2196 ± 1127	1466 ± 470.3	3868 ± 593.1	2894 ± 1109	152.7 ± 79.47	2205 ± 890.8	**4276 ± 357.1**
`Ant-v4`	6046 ± 552.5	4644 ± 966.2	976.9 ± 302.6	5645 ± 914	5908 ± 811.3	3652 ± 1685	2245 ± 761.4	**6048 ± 464.4**
`Humanoid-v4`	5645 ± 904.1	1715 ± 783.4	2023 ± 604.9	5241 ± 302.4	5286 ± 703.2	160.2 ± 32.99	1814 ± 465.6	**5668 ± 339.5**
`button-press-v2`	**1436 ± 1241**	919.6 ± 1066	1093 ± 1016	635.5 ± 566	1366 ± 1133	1128 ± 886.1	1424 ± 898.3	835 ± 1190
`door-open-v2`	3671 ± 1606	**4288 ± 1431**	2691 ± 1415	3784 ± 809.1	4232 ± 454.2	3956 ± 941	3818 ± 1258	4083 ± 1059
`drawer-close-v2`	**4839 ± 1726**	4706 ± 1578	3880 ± 1916	4743 ± 1718	3039 ± 2206	4750 ± 1560	4178 ± 1788	4808 ± 1324
`drawer-open-v2`	2762 ± 1212	**4074 ± 1421**	2510 ± 1467	2951 ± 1511	2617 ± 1642	2256 ± 1819	2820 ± 1463	2048 ± 1187
`peg-insert-side-v2`	1226 ± 1769	431.4 ± 1417	402.4 ± 1855	432 ± 1191	7.378 ± 2.552	317.8 ± 579.6	339.8 ± 1211	**2146 ± 1752**
`pick-place-v2`	12.49 ± 110.5	212.5 ± 355	**509.3 ± 672.7**	7.501 ± 478.7	5.553 ± 2.761	16.98 ± 378	451.2 ± 847.2	6.942 ± 732.7
`push-v2`	191.1 ± 1282	310.9 ± 1005	**777.9 ± 928.9**	42.5 ± 930.5	25.99 ± 76.78	415.2 ± 971.5	74.32 ± 847.7	729.7 ± 1571
`reach-v2`	1746 ± 1127	2889 ± 909.7	2992 ± 1466	2092 ± 1304	**3390 ± 1243**	2552 ± 1066	3068 ± 1231	2735 ± 1113
`window-open-v2`	2668 ± 1515	2501 ± 1764	3029 ± 1476	3452 ± 1150	491.3 ± 1326	**3916 ± 1023**	2294 ± 1551	2786 ± 1488
`window-close-v2`	4404 ± 712	4520 ± 214.5	4352 ± 342.3	4202 ± 1145	4372 ± 704.6	**4574 ± 866.3**	4022 ± 1191	4291 ± 556
`basketball-v2`	780.9 ± 1238	602.4 ± 1106	809.1 ± 901.4	627.9 ± 648.4	9.246 ± 197.2	1126 ± 1670	**1972 ± 1197**	524.4 ± 702.6
`dial-turn-v2`	1214 ± 952.6	1552 ± 1250	**2385 ± 1486**	1228 ± 1206	2192 ± 1593	1446 ± 1020	2157 ± 1465	1258 ± 1034
`sweep-into-v2`	**4237 ± 1348**	1417 ± 1448	3007 ± 1677	1977 ± 1907	530.2 ± 1712	3650 ± 1591	2203 ± 2026	1496 ± 2042
`hammer-v2`	**4308 ± 1215**	3500 ± 1344	2664 ± 1477	1340 ± 1099	487.5 ± 945	2812 ± 1869	3620 ± 1311	2866 ± 1886
`assembly-v2`	**2606 ± 660.4**	1941 ± 1241	2265 ± 861.8	2085 ± 801.6	947.2 ± 983.3	439.6 ± 1429	2376 ± 1032	1528 ± 921.1
`BipedalWalker-v3`	315.2 ± 12.26	325.4 ± 8.892	218.6 ± 115.9	321.2 ± 2.343	321.1 ± 21.67	**331.9 ± 6.717**	247.4 ± 72.5	322.1 ± 6.48
`BipedalWalkerHardcore-v3`	**189 ± 86.95**	152.6 ± 52.39	-71.08 ± 30.12	18.27 ± 45.01	-32.19 ± 56.1	114.1 ± 65.9	-84.9 ± 17.95	53.3 ± 40.58
DeepMind Control Aggregate	0.6191	**0.6326**	0.5823	0.4588	0.293	0.5003	0.5787	0.5352
MuJoCo Aggregate	0.818	0.4663	0.3077	0.7716	0.6304	0.1423	0.4079	**0.8305**
MetaWorld Aggregate	**0.209**	0.193	0.1904	0.149	0.09027	0.1881	0.2017	0.1867
Box2D Aggregate	**0.9391**	0.8911	0.319	0.7204	0.6658	0.8659	0.3764	0.7679
All Aggregate	**0.428**	0.3709	0.3026	0.3358	0.226	0.2802	0.3255	0.39



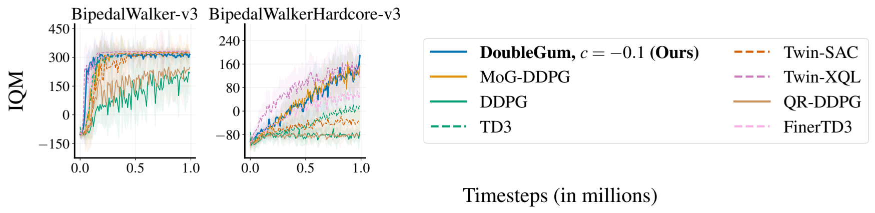
(a) DeepMind Control



(b) MuJoCo



(c) MetaWorld

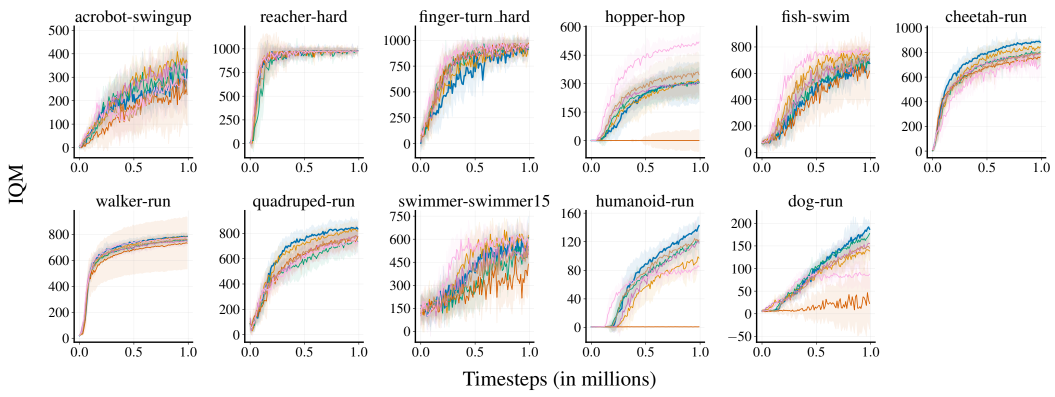


(d) Box2D

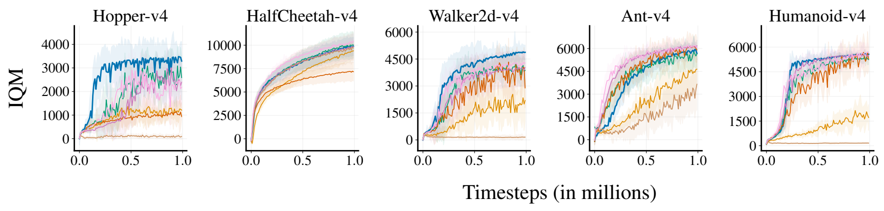
Figure 18: Continuous control with default pessimism hyperparameters, per-task IQM ± standard deviation. Methods that default to use Twin Networks are dashed. The legend for all graphs is in Figure 18d.

Table 8: Continuous control with the best pessimism hyperparameters adjusted per suite after 1M timesteps

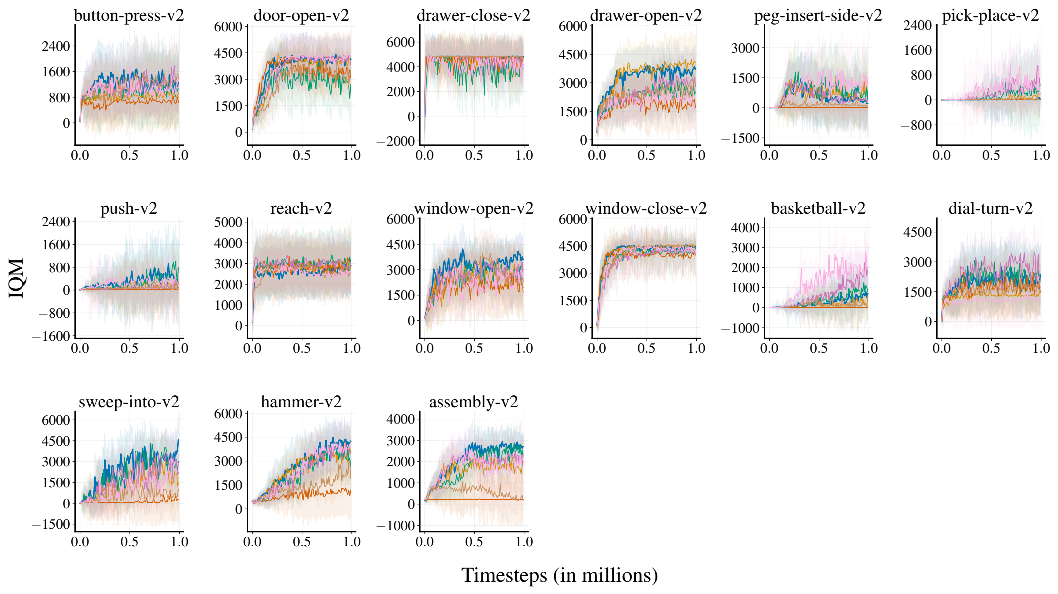
Suite/Task	Score at 1M timesteps (IQM $\pm$ standard deviation where appropriate) with best pessimism						
	DoubleGum	MoG-DDPG	DDPG/TD3	SAC	XQL	QR-DDPG	FinerTD3
`acrobot-swingup`	330.2 $\pm$ 79.44	**364.9** $\pm$ **71.75**	334.7 $\pm$ 96.92	226.6 $\pm$ 134.5	356.7 $\pm$ 88.26	225.7 $\pm$ 131.1	283.3 $\pm$ 89.79
`reacher-hard`	**979.4** $\pm$ **54.28**	976.9 $\pm$ 4.985	975.7 $\pm$ 26.78	976.2 $\pm$ 56.58	972.9 $\pm$ 14.86	975.4 $\pm$ 13.21	976.3 $\pm$ 27.78
`finger-turn_hard`	931.8 $\pm$ 98.91	935.4 $\pm$ 77.4	909.1 $\pm$ 50.68	**972.4** $\pm$ **25.53**	924.3 $\pm$ 73.62	952.6 $\pm$ 65.92	956.4 $\pm$ 74.46
`hopper-hop`	306.7 $\pm$ 101	313.3 $\pm$ 99.74	305.7 $\pm$ 80.12	0.007651 $\pm$ 56.18	304.3 $\pm$ 69.23	346.8 $\pm$ 105	**519.5** $\pm$ **47.02**
`fish-swim`	675.7 $\pm$ 58.5	758.1 $\pm$ 62.68	710.2 $\pm$ 94.54	616.4 $\pm$ 226.4	695.2 $\pm$ 54.08	708.8 $\pm$ 38.73	**761.6** $\pm$ **41.15**
`cheetah-run`	**883.1** $\pm$ **22.51**	844.9 $\pm$ 53.54	804.5 $\pm$ 64.31	764.5 $\pm$ 74.55	785.9 $\pm$ 63.43	791.1 $\pm$ 69.23	699 $\pm$ 84.17
`walker-run`	**783.6** $\pm$ **26.97**	778.7 $\pm$ 21.86	755.6 $\pm$ 28.29	732 $\pm$ 199.4	743.9 $\pm$ 29.61	759.8 $\pm$ 24.46	777 $\pm$ 12.7
`quadruped-run`	**835.1** $\pm$ **65.11**	818.2 $\pm$ 76.37	736 $\pm$ 87.58	774 $\pm$ 62.96	772.4 $\pm$ 80.69	775 $\pm$ 54.76	724.4 $\pm$ 92.74
`swimmer-swimmer15`	608.9 $\pm$ 137.8	**623.4** $\pm$ **81.98**	531.5 $\pm$ 104.1	437.8 $\pm$ 192.5	499.8 $\pm$ 64.12	552.7 $\pm$ 140.2	560.2 $\pm$ 106.4
`humanoid-run`	**142.5** $\pm$ **12.6**	96.97 $\pm$ 12.58	119.6 $\pm$ 25.09	0.9647 $\pm$ 0.1496	117.8 $\pm$ 17.68	122.4 $\pm$ 9.644	85.08 $\pm$ 6.825
`dog-run`	**187.1** $\pm$ **13.14**	138.8 $\pm$ 20.58	177.4 $\pm$ 12.17	22.63 $\pm$ 56.75	155.5 $\pm$ 22.55	145.3 $\pm$ 28.37	87.51 $\pm$ 32.47
`Hopper-v4`	**3290** $\pm$ **829.1**	1211 $\pm$ 319.2	2589 $\pm$ 939.2	942 $\pm$ 135.3	2398 $\pm$ 946.4	94.89 $\pm$ 122.7	2183 $\pm$ 890.6
`HalfCheetah-v4`	9874 $\pm$ 997.4	9398 $\pm$ 1044	10020 $\pm$ 1390	7171 $\pm$ 1000	9855 $\pm$ 1323	9694 $\pm$ 1571	**10420** $\pm$ **328.8**
`Walker2d-v4`	**4871** $\pm$ **403.7**	2196 $\pm$ 1127	3868 $\pm$ 593.1	2894 $\pm$ 1109	4027 $\pm$ 962	152.7 $\pm$ 79.47	4276 $\pm$ 357.1
`Ant-v4`	5681 $\pm$ 416.8	4644 $\pm$ 966.2	5645 $\pm$ 914	5908 $\pm$ 811.3	**6185** $\pm$ **168.3**	3652 $\pm$ 1685	6048 $\pm$ 464.4
`Humanoid-v4`	5565 $\pm$ 160.7	1715 $\pm$ 783.4	5241 $\pm$ 302.4	5286 $\pm$ 703.2	5452 $\pm$ 243.1	160.2 $\pm$ 32.99	**5668** $\pm$ **339.5**
`button-press-v2`	1436 $\pm$ 1241	919.6 $\pm$ 1066	1093 $\pm$ 1016	645.5 $\pm$ 728.3	1424 $\pm$ 898.3	**1506** $\pm$ **1047**	1086 $\pm$ 758.2
`door-open-v2`	3671 $\pm$ 1606	**4288** $\pm$ **1431**	2691 $\pm$ 1415	3114 $\pm$ 1166	3818 $\pm$ 1258	3512 $\pm$ 1387	4268 $\pm$ 953.8
`drawer-close-v2`	**4839** $\pm$ **1726**	4706 $\pm$ 1578	3880 $\pm$ 1916	4749 $\pm$ 1339	4178 $\pm$ 1788	4588 $\pm$ 2042	3741 $\pm$ 2037
`drawer-open-v2`	2762 $\pm$ 1212	**4074** $\pm$ **1421**	2510 $\pm$ 1467	1710 $\pm$ 1514	2820 $\pm$ 1463	2972 $\pm$ 1382	1608 $\pm$ 1333
`peg-insert-side-v2`	**1226** $\pm$ **1769**	431.4 $\pm$ 1417	402.4 $\pm$ 1855	9.693 $\pm$ 232.9	339.8 $\pm$ 1211	100.8 $\pm$ 1462	1092 $\pm$ 1699
`pick-place-v2`	12.49 $\pm$ 110.5	212.5 $\pm$ 355	**509.3** $\pm$ **672.7**	4.821 $\pm$ 1.873	451.2 $\pm$ 847.2	26.68 $\pm$ 47.73	501.4 $\pm$ 963.9
`push-v2`	191.1 $\pm$ 1282	310.9 $\pm$ 1005	**777.9** $\pm$ **928.9**	90.81 $\pm$ 745	74.32 $\pm$ 847.7	76.38 $\pm$ 95.72	256.8 $\pm$ 453.4
`reach-v2`	1746 $\pm$ 1127	2889 $\pm$ 909.7	2992 $\pm$ 1466	2974 $\pm$ 1393	**3068** $\pm$ **1231**	2583 $\pm$ 1345	2402 $\pm$ 922.9
`window-open-v2`	2668 $\pm$ 1515	2501 $\pm$ 1764	**3029** $\pm$ **1476**	1957 $\pm$ 1832	2294 $\pm$ 1551	1672 $\pm$ 1813	2964 $\pm$ 1365
`window-close-v2`	4404 $\pm$ 712	4520 $\pm$ 214.5	4352 $\pm$ 342.3	4000 $\pm$ 1045	4022 $\pm$ 1191	**4528** $\pm$ **232.3**	4272 $\pm$ 260
`basketball-v2`	780.9 $\pm$ 1238	602.4 $\pm$ 1106	809.1 $\pm$ 901.4	9.932 $\pm$ 11.24	**1972** $\pm$ **1197**	99.83 $\pm$ 745.6	1868 $\pm$ 959.1
`dial-turn-v2`	1214 $\pm$ 952.6	1552 $\pm$ 1250	**2385** $\pm$ **1486**	1472 $\pm$ 1178	2157 $\pm$ 1465	1690 $\pm$ 1408	1101 $\pm$ 1255
`sweep-into-v2`	**4237** $\pm$ **1348**	1417 $\pm$ 1448	3007 $\pm$ 1677	247.9 $\pm$ 1718	2203 $\pm$ 2026	1008 $\pm$ 1782	2844 $\pm$ 1934
`hammer-v2`	**4308** $\pm$ **1215**	3500 $\pm$ 1344	2664 $\pm$ 1477	1180 $\pm$ 1650	3620 $\pm$ 1311	1892 $\pm$ 1387	4151 $\pm$ 1115
`assembly-v2`	**2606** $\pm$ **660.4**	1941 $\pm$ 1241	2265 $\pm$ 861.8	218.1 $\pm$ 492.3	2376 $\pm$ 1032	164 $\pm$ 401.5	2067 $\pm$ 1507
`BipedalWalker-v3`	315.2 $\pm$ 12.26	325.4 $\pm$ 8.892	321.2 $\pm$ 2.343	321.1 $\pm$ 21.67	311.3 $\pm$ 18.27	**331.9** $\pm$ **6.717**	322.1 $\pm$ 6.48
`BipedalWalkerHardcore-v3`	**189** $\pm$ **86.95**	152.6 $\pm$ 52.39	18.27 $\pm$ 45.01	-32.19 $\pm$ 56.1	-31.74 $\pm$ 18.9	114.1 $\pm$ 65.9	53.3 $\pm$ 40.58
DMC Aggregate	0.6191	**0.6326**	0.5823	0.4762	0.5918	0.5787	0.6121
MuJoCo Aggregate	**0.8367**	0.4663	0.7716	0.6304	0.1423	0.8061	0.8305
MetaWorld Aggregate	**0.2505**	0.193	0.1904	0.08515	0.1204	0.2017	0.2042
Box2D Aggregate	**0.9391**	0.8911	0.7204	0.6658	0.8659	0.601	0.7679
All Aggregate	**0.4707**	0.3709	0.3979	0.2719	0.2786	0.3924	0.4239



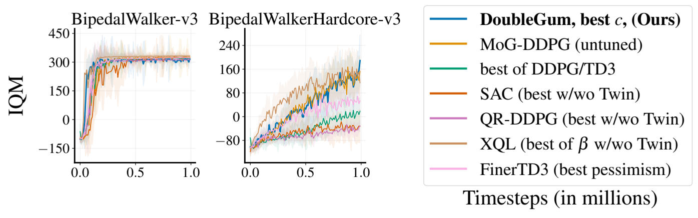
(a) DeepMind Control



(b) MuJoCo



(c) MetaWorld



(d) Box2D

Figure 19: Continuous control with the best pessimism hyperparameters adjusted per suite, per-task IQM $\pm$ standard deviation. The legend for all graphs is in Figure 19d.