CAQL: Continuous Action Q-Learning

Moonkyung Ryu*, Yinlam Chow*, Ross Anderson, Christian Tjandraatmadja, Craig Boutilier
Google Research
{mkryu,yinlamchow,rander,ctjandra,cboutilier}@google.com

Abstract

Reinforcement learning (RL) with value-based methods (e.g., Q-learning) has shown success in a variety of domains. One challenge in applying Q-learning to *continuous-action* RL problems, however, is the continuous action maximization (*max-Q*) required for optimal Bellman backup. While it is common to restrict the parameterization of the Q-function to be concave in actions to simplify the max-Q problem, such a restriction might lead to performance degradation. Alternatively, when the Q-function is parameterized with a generic feed-forward neural network (NN), the max-Q problem can be NP-hard. In this work, we propose a CAQL framework which minimizes the Bellman residual using Q-learning with one of several *plug-and-play* action optimizers. In particular, leveraging the strides of optimization theories in deep NN, we show that max-Q problem can be solved optimally with *mixed-integer programming (MIP)*—when the Q-function has sufficient representation power, this MIP-based optimization induces better policies and is more robust than counterparts, e.g., CEM or GA, that approximate the max-Q solution. To speed up inference of CAQL, we introduce the *action function* that concurrently learns the optimal policy. To demonstrate the efficiency of CAQL we compare it with state-of-the-art RL algorithms on benchmark continuous control problems that have different degrees of action constraints and show that CAQL significantly outperforms policy-based methods in heavily constrained environments.

1 Introduction

Reinforcement learning (RL) has shown success in a variety of domains such as games (Mnih et al., 2013) and recommender systems (RSs) (Gauci et al., 2018). When the action space is finite, value-based algorithms such as Q-learning (Watkins & Dayan, 1992), which implicitly finds a policy by learning the optimal value function, are often very efficient because action optimization can be done by exhaustive enumeration. By contrast, in problems, such as in robotics (Peters & Schaal, 2006), with a continuous action space, policy-based algorithms, such as policy gradient (PG) (Sutton et al., 2000; Silver et al., 2014) or cross-entropy policy search (CEPS) (Mannor et al., 2003; Kalashnikov et al., 2018), which directly learn a return-maximizing policy, have proven more practical. Recently, methods such as ensemble critic (Fujimoto et al., 2018) and entropy regularization (Haarnoja et al., 2018) have been developed to improve the performance of policy-based RL algorithms.

Policy-based approaches require a reasonable choice of policy parameterization. In some continuous control problems, Gaussian distributions over actions conditioned on some state representation is used. However, in applications such as recommender systems (RSs), where actions often take form of high-dimensional item-feature vectors, policies cannot typically be modeled by common action distributions. Furthermore, the admissible action set in RL is constrained in practice, for example, when actions must lie within a specific range for safety (Chow et al., 2018). In RSs, the admissible actions are random functions of the state (Boutilier et al., 2018). In such cases, it is non-trivial to define policy parameterizations that handle such factors. On the other hand, value-based algorithms are well-suited to these settings, providing potential advantage over policy methods. Moreover, at least with linear function approximations (Melo & Ribeiro, 2007), under reasonable assumptions, Q-learning has been shown to converge to optimality, while such optimality guarantees for non-convex policy-based methods are generally limited (Fazel et al., 2018). Empirical results also suggest that

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

value-based methods are more data-efficient and less sensitive to hyper-parameter tuning (Quillen et al., 2018). Of course, when the action space is large, exhaustive action enumeration in value-based algorithms can be expensive—-one solution is to represent actions with continuous features (Dulac-Arnold et al., 2015).

The main challenge in applying value-based algorithms to continuous-action domains is selecting optimal actions. Previous work in this direction falls into three broad categories. The first solves the inner maximization of the (optimal) Bellman residual loss using global nonlinear optimizers, such as the cross-entropy method (CEM) for QT-Opt (Kalashnikov et al., 2018), gradient ascent (GA) for Actor-Expert (Lim et al., 2018), and action discretization (Uther & Veloso, 1998; Smart & Kaelbling, 2000; Lazaric et al., 2008). However, these approaches do not guarantee optimality. The second approach restricts the Q-function parameterization so that the optimization problem is tractable. For instance, wire-fitting (Gaskett et al., 1999; III & Klopf, 1993) approximates Q-values piecewise-linearly over a discrete set of points, chosen to ensure the maximum action is one of the extreme points. The normalized advantage function (NAF) (Gu et al., 2016) constructs the state-action advantage function to be quadratic, hence analytically solvable. Parameterizing the Q-function with an input-convex neural network (ICNN) (Amos et al., 2017) ensures it is concave. These restricted functional forms, however, may lead to performance degradation if the domain does not conform to the imposed structure. The third category replaces optimal Q-values with a "soft" counterpart (Haarnoja et al., 2018): an entropy regularization term ensures that both the optimal Q-function and the optimal policy have closed-form solutions. The Q-update is typically approximated by sampling estimates, which can be computationally expensive. Moreover, the sub-optimality gap of this soft policy scales with the interval of action space (Neu et al., 2017), which grows with the dimensionality of action space.

Motivated by the shortcomings of the prior approaches, we propose *Continuous Action Q-learning (CAQL)*, a Q-learning framework for continuous actions in which the Q-function is modeled by a *generic* feed-forward neural network. [1] Our contribution is two-fold. The CAQL method minimizes the Bellman residual using Q-learning, using one of several "plug-and-play" action optimizers. We show that "max-Q" optimization can be formulated as a mixed-integer programming (MIP) that solves max-Q optimally—when the Q-function has sufficient representation power, we demonstrate that MIP-based optimization induces better policies and is more robust than counterparts, CEM or GA, that approximate the max-Q solution. Second, we compare CAQL with several state-of-the-art RL algorithms on several benchmark continuous-control problems with varying degrees of action constraints. Value-based CAQL is generally competitive, and outperforms policy-based methods in heavily constrained environments, sometimes significantly.

2 Preliminaries

We consider an infinite-horizon, discounted Markov decision process (Puterman, 2014) with states X, (continuous) action space A, reward function R, a transition kernel P, initial state distribution β and discount factor $\gamma \in [0, 1)$, all having the usual meaning. A (stationary, Markovian) policy π specifies a distribution $\pi(\cdot|x)$ over actions to be taken at state x. Let Δ be the set of such policies. The *expected cumulative return* of $\pi \in \Delta$ is $J(\pi) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid P, R, x_0 \sim \beta, \pi]$. An optimal policy π^* satisfies $\pi^* \in \arg\max_{\pi \in \Delta} J(\pi)$.

The Bellman operator over Q-functions Q (state-action value functions), $F[Q](x, a) = R(x, a) + \gamma \sum_{x' \in X} P(x'|x, a) \max_{a' \in A} Q(x', a')$, has unique fixed point $Q^*(x, a)$ (Puterman, 2014), which is the optimal Q-function $Q^*(x, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(x_t, a_t) \mid x_0 = x, a_0 = a, \pi^*\right]$. An optimal (deterministic) policy π^* can be extracted from Q^*: $\pi^*(a|x) = \mathbf{1}\{a = a^*(x)\}$, where $a^*(x) \in \arg\max_a Q^*(x, a)$.

For large or continuous state/action spaces, the optimal Q-function can be approximated, e.g., using a deep NN as in DQN (Mnih et al., 2013). In DQN, the value function Q_θ is updated using the value label $r + \gamma \max_{a'} Q_{\theta^{\text{target}}}(x', a')$, where $Q_{\theta^{\text{target}}}$ is a *target* Q-function. Instead of training these weights jointly, θ^{target} is updated in a separate iterative fashion using the previous θ for a fixed number of training steps, or by averaging $\theta^{\text{target}} \leftarrow \tau\theta + (1 - \tau)\theta^{\text{target}}$ for some small momentum weight $\tau \in [0, 1]$ (Mnih et al., 2016). DQN is *off-policy*—the target is valid no matter how the experience was generated. Typically, the loss is minimized over mini-batches of past data $(x, a, r, x') \in B$ sampled from an experience replay buffer B (Lin &

[1] Result can be extended to handle convolutional NNs, but is omitted for brevity.

Mitchell, 1992). One common loss function for training Q_{θ^*} is *mean squared Bellman error*: $\min_\theta \sum_{i=1}^{|B|} \left(Q_\theta(x_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta^{\text{target}}}(x_i', a')\right)^2$. Under this loss, RL can be viewed as ℓ_2-regression of $Q_\theta(\cdot,\cdot)$ w.r.t. target labels $r + \gamma \max_{a'} Q_{\theta^{\text{target}}}(x', a')$. We augment DQN, using *double Q-learning* for more stable training (Hasselt et al., 2016), whose loss is:

$$\min_\theta \sum_{i=1}^{|B|} \left(Q_\theta(x_i, a_i) - r_i - \gamma Q_{\theta^{\text{target}}}(x_i', \arg\max_{a'} Q_\theta(x_i', a'))\right)^2. \tag{1}$$

A *hinge loss* can also be used in Q-learning, and has connections to the linear programming (LP) formulation of MDP (Puterman (2014)). The weights of the optimal Q-network can be specified as: $\min_\theta \frac{1}{|B|} \sum_{i=1}^{|B|} Q_\theta(x_i, a_i) + \lambda \left(r_i + \gamma \max_{a' \in A} Q_\theta(x_i', a') - Q_\theta(x_i, a_i)\right)_+$, where $\lambda > 0$ is a tunable penalty weight w.r.t. constraint: $r + \gamma \max_{a' \in A} Q_\theta(x', a') \le Q_\theta(x, a)$, $\forall (x, a, r, x') \in B$. To stabilize training, as above, we replace the Q-network of the inner-maximization with the target Q-network and the optimal Q-value with the double-Q label, giving (see Appendix A for details):

$$\min_\theta \frac{1}{|B|} \sum_{i=1}^{N} Q_\theta(x_i, a_i) + \lambda \left(r_i + \gamma Q_{\theta^{\text{target}}}(x_i', \arg\max_{a'} Q_\theta(x_i', a')) - Q_\theta(x_i, a_i)\right)_+. \tag{2}$$

In this work we assume the Q-function approximation Q_θ to be a feed-forward network (MLP). Specifically, let Q_θ be a K-layer feed-forward NN with state-action input (x, a) (where a is in a d-dimensional real vector space) and hidden layers arranged according to the equations:

$$z_1 = (x, a),\ \hat{z}_j = W_{j-1} z_{j-1} + b_{j-1},\ z_j = h(\hat{z}_j),\ j = 2, \ldots, K,\ Q_\theta(x, a) := c^\top \hat{z}_K,^2 \tag{3}$$

where (W_j, b_j) are the multiplicative and bias weights, c is the output weight of the Q-network, $\theta = \left(c, \{(W_j, b_j)\}_{j=1}^{K-1}\right)$ are the weights of the Q-network, $\hat{z}_j$ denotes pre-activation values at layer j, and $h(\cdot)$ is the (component-wise) activation function. For simplicity, in the following analysis we restrict our attention to the case when the activation functions are ReLU's. We also assume that the action space A is a d-dimensional ℓ_∞-ball $B_\infty(\overline{a}, \Delta)$ with some radius $\Delta > 0$ and center $\overline{a}$.

3 Continuous Action Q-Learning Algorithm

Policy-based methods (Silver et al., 2014; Fujimoto et al., 2018; Haarnoja et al., 2018) have been widely-used to handle continuous actions in RL. However, they suffer from several well-known difficulties, e.g., (i) modeling high-dimensional action distributions, (ii) handling action constraints, and (iii) data-inefficiency. Motivated by earlier work on value-based RL methods, such as QT-Opt (Kalashnikov et al., 2018) and Actor-Expert (Lim et al., 2018), we propose *Continuous Action Q-learning (CAQL)*, a general framework for continuous-action value-based RL, in which the Q-function is parameterized by a NN (Eq. 3). One novelty of CAQL is the formulation of the "max-Q" problem, i.e., the inner maximization in (1) and (2), as a mixed-integer programming (MIP).

The benefit of the MIP formulation is that it guarantees that we find the optimal action (and its true bootstrapped Q-value) when computing target labels (and at inference time). We show empirically that this can induce to better performance, especially when the Q-network has sufficient representation power. Moreover, since a MIP can readily model a wide range of combinatorial constraints (e.g., routing, scheduling, packing), it offers considerable flexibility to incorporate complex action constraints in RL. That said, solving MIPs to optimality is computationally intensive. To alleviate this, we develop methods to systematically reduce the computational demands of the inner maximization—at the price of some approximation. In Sec. 3.2, we introduce the *action function* to approximate the $\arg\max$-policy at inference time.

3.1 Plug-N-Play Max-Q Optimizers

When the Q-function is parameterized with a feed-forward ReLU NN (see Eq. (3)), the inner maximization in CAQL is generally non-convex and NP-hard (Tjeng et al., 2017). In this section, we illustrate how this problem can be formulated as a MIP, which can be solved using off-the-shelf optimization packages (e.g., SCIP (Gleixner et al., 2018), CPLEX (CPLEX, 2019), Gurobi (Gurobi, 2019)). In addition, we detail how approximate optimizers, specifically, gradient ascent (GA) and the cross-entropy method (CEM), can trade optimality for speed in max-Q computation within CAQL.

[2] Without loss of generality, we simplify the NN by omitting output bias and output activation function.

Mixed-Integer Programming (MIP) A trained feed-forward ReLU neural network can be modeled via mixed-integer programming by formulating the nonlinear activation function at each individual neuron with binary constraints. Specifically, for a ReLU with pre-activation function of form $z = \max\{0, w^\top x + b\}$, where $x \in [L, U]$ is a d-dimensional bounded input, $w \in \mathbb{R}^d$, $b \in \mathbb{R}$, and $L, U \in \mathbb{R}^d$ are the weights, bias and lower-upper bounds respectively, consider the following set with a binary variable ζ indicating whether the ReLU is active or not:

$$R(w, b, L, U) = \left\{ (x, z, \zeta) \;\middle|\; \begin{array}{c} z \geq w^\top x + b,\ z \geq 0,\ z \leq w^\top x + b - M^-(1-\zeta),\ z \leq M^+\zeta, \\ (x, z, \zeta) \in [L, U] \times \mathbb{R} \times \{0, 1\} \end{array} \right\}.$$

In this formulation both $M^+ = \max_{x \in [L,U]} w^\top x + b$ and $M^- = \min_{x \in [L,U]} w^\top x + b$ can be computed in linear time in d. These constraints ensure that z is the output of the ReLU: If $\zeta = 0$, then they are reduced to $z = 0 \geq w^\top x + b$, and if $\zeta = 1$, then they become $z = w^\top x + b \geq 0$.

Extending this to model a multi-layer perceptron, the ReLU neural network in (3) can be constructed by *chaining* copies of intermediate ReLU networks. Since the output layer is linear, the MIP objective is linear as well. More precisely, suppose the ReLU Q-network has m_j neurons in each layer $j \in \{2, \ldots, K\}$, then for any given state $x \in X$, the inner maximization problem $\max_a Q_\theta\ (x, a)$ can be reformulated as the following MIP:

$$\begin{array}{ll} \max & c^\top z_K \\ \text{s.t.} & z_1 := a \in B_\infty(\overline{a}, \Delta), \\ & (z_{j-1}, z_{j,i}, \zeta_{j,i}) \in R(W_{j,i}, b_{j,i}, L_{j-1}, U_{j-1}),\ j \in \{2, \ldots, K\},\ i \in \{1, \ldots, m_j\}, \end{array}$$

where $L_1 = \overline{a} - \Delta, U_1 = \overline{a} + \Delta$ are the (action) input-bound vectors. Here, $W_{j,i} \in \mathbb{R}^{m_j}$ and $b_{j,i} \in \mathbb{R}$ are the weights and bias of the neuron i in layer j. Furthermore, L_j, U_j are interval bounds for the outputs of the neurons in layer j for $j \geq 2$, and computing them can be done via interval arithmetic or other propagation methods (Weng et al., 2018) from the initial action space bounds. As detailed by Anderson et al. (2019), this can be further tightened with additional constraints, and its implementation can be found in the *tf.opt* package described therein. We emphasize that the MIP returns *provably global optima*, unlike GA and CEM. Even when interrupted with stopping conditions such as a time limit, MIP often produces high-quality solutions in practice.

We note that Say et al. (2017) proposed a MIP formulation to solve a planning problem using a non-linear state transition dynamics model learned by a neural network. While related, it differs from the max-Q problem addressed here.

Gradient Ascent GA (Nocedal & Wright, 2006) is a simple first-order optimization method for finding the (local) optimum of a differentiable objective function, such as a neural network Q-function. At any state $x \in X$, given an initial action a_0, the optimal action $\arg\max_a Q_\theta\ (x, a)$ is computed iteratively by $a_{t+1} \leftarrow a_t + \eta \nabla_a Q_\theta(x, a)$, where $\eta > 0$ is a step size (either a tunable parameter or computed using back-tracking line search (Nocedal & Yuan, 1998)). The process GA repeats until convergence, $|Q_\theta(x, a_{t+1}) - Q_\theta(x, a_t)| < \epsilon$, or a maximum iteration count is reached.

Cross-Entropy Method CEM (Rubinstein, 1999) is a derivative-free optimization algorithm. At any given state $x \in X$, it samples a batch of N actions $\{a_i\}_{i=1}^N$ from A using a fixed distribution, e.g., a Gaussian, and ranks the corresponding Q-values $\{Q_\theta(x, a_i)\}_{i=1}^N$. Using the top $K < N$ actions, it then updates the sampling distribution, e.g., using the sample mean and covariance to update the Gaussian. This is repeated until convergence or a maximum iteration count is reached.

3.2 Action Function

In traditional Q-learning, the policy is "implemented" by acting greedily w.r.t. the learned Q-function: $\pi^*(x) = \arg\max_a Q_\theta\ (x, a)$.[3] However, computing the optimal action can be expensive in the continuous case, which may be especially problematic at inference time (e.g., when computational power is limited in, say embedded systems, or real-time response is critical). To mitigate the problem, we can use an *action function* $\pi_w : X \to A$, effectively a trainable actor network, to approximate the greedy-action mapping π^* w.r.t. Q_θ. We train π_w as follows. Training data takes the form $\{(x_i, q_i^*)\}_{i=1}^{|B|}$, where $q_i^* = \max_a Q_\theta\ (x_i, a)$ is the max-Q label at state x_i. Action function learning is then simply a supervised regression problem:

[3] Some exploration strategy may be incorporated as well.

$w^* \in \arg\min_w \sum_{i=1}^{|B|} (q_i^* - Q_\theta(x_i, \pi_w(x_i)))^2$. This is similar to the notion of "distilling" an optimal policy from max-Q labels, as considered in Actor-Expert (Lim et al., 2018). Unlike that approach, in which a separate stochastic policy network is jointly learned with the Q-function to maximize the likelihood with the underlying optimal policy, our method simply learns a deterministic state-to-action mapping to approximate $\arg\max_a Q_\theta(x, a)$—this does not require distribution matching and will generally be more stable. We note that the use of the action function in CAQL is an (optional) computational convenience to accelerate data collection (e.g., in simulation) and inference.

4 Experiments on MuJoCo Benchmarks

To illustrate the effectiveness of the CAQL algorithms, in this section we compare their performance with that from several state-of-the-art baseline methods on multiple domains. Regarding the choices of baseline methods, we compare with (i) DDPG (Silver et al., 2014), (ii) TD3 (Fujimoto et al., 2018), which are two popular policy-based deep RL algorithms, (iii) NAF (Gu et al., 2016), which is a value-based method with action-quadratic Q-function. We also train CAQL with three different max-Q optimizers, i.e., MIP, GA, and CEM, where the CAQL-CEM counterpart resembles QT-Opt (Kalashnikov et al., 2018), and the CAQL-GA counterpart resembles Actor-Expert (Lim et al., 2018), in order to investigate the amount of performance improvement that learning with optimal Bellman residual via MIP (which corresponds to DDQN (Hasselt, 2010) with continuous actions) provides over approximations with GA and CEM at the cost of additional computation overhead.

We evaluate CAQL on one classical control benchmark (i.e., Pendulum) and five challenging MuJoCo benchmarks (i.e., Hopper, Walker2D, HalfCheetah, Ant, Humanoid). Different from most previous experiments (for example that in the aforementioned work), in the following not only do we evaluate the RL algorithms on domains with default action ranges, but also on the same domains with smaller action ranges (see Table 3 in Appendix C for all the action ranges used in our experiments). [4] The motivation of this setting is two-fold: (i) To simulate real-world problems (Dulac-Arnold et al., 2019), where the restricted ranges represent the safe/constrained action sets; (ii) To validate the hypothesis that action-distribution learning in policy-based methods is inept in handling constraints, while CAQL does not have this issue. Considering the computational overhead of MIP, we reduce the episode limit from 1000 steps to 200 steps and use a smaller neural network. Both changes would lead to lower returns than that reported in existing RL experiments (Duan et al., 2016). Details on the NN architecture and hyper parameters are described in Appendix C.

Performance evaluation is done periodically (once per 1000 training iterations) with a policy that does not include exploration. Each measurement is an average of 10 episodic return values, each generated from a separate random seed. To smoothen the learning curves, data points are averaged over a sliding window of size 3. Similar to the setting in Actor-Expert (Lim et al., 2018), for efficient evaluation the CAQL measurements are based on trajectories that are generated by following the learned action function instead of following the optimal action w.r.t. the learned Q-function.

Table 1 shows 95-percentile for the mean and standard deviation of the final returns using the *best* hyper-parameter setting. CAQL significantly outperforms NAF for most benchmarks, as well as DDPG and TD3 on 10 out of 14 benchmarks. Over all the CAQL policies, the ones trained with MIP are among the top-performing ones on all the experiments except Ant [-0.25, 0.25] and Humanoid [-0.25, 0.25]. This verifies our earlier conjecture about CAQL that learning with optimal Bellman residual will have better performance whenever the Q-function has sufficient representation power (which is more likely in low dimensional tasks). Additionally, the performance of CAQL-MIP policy has slightly lower variance than that trained with GA and CEM for most benchmarks. Table 2 shows 95-percentile for the mean and standard deviation of final returns over all 320 configurations (32 hyper parameter combinations $\times$ 10 random seeds). This result illustrates the sensitivity to hyper parameters in each method. CAQL has the best performance on 13 out of 14 tasks, and in particular policies trained with MIP are the best on 8 out of 14 tasks. This corroborates with the hypothesis that value-based methods are generally more robust to hyper parameters than policy-based counterparts.

However, worth-noting that CAQL-MIP suffers from performance degradation in several high dimensional environments with large action ranges (e.g., Ant [-0.25, 0.25] and Humanoid [-0.25, 0.25]). In these experiments its performance is even worse than that of CAQL-GA or CAQL-CEM.

[4] Smaller action ranges usually corresponds to easier MIP problems in the max-Q computation. Unfortunately due to high complexity of MIP in more complex environments such as Walker2D, HalfCheetah, Ant, and Humanoid, we only run experiments with smaller action ranges than the default values.

Environment [Action range]	CAQL-MIP	CAQL-GA	CAQL-CEM	NAF	DDPG	TD3
Pendulum [-0.66, 0.66]	**-339.5** ± 158.3	-342.4 ± 151.6	-394.6 ± 246.5	-449.4 ± 280.5	-407.3 ± 180.3	-437.5 ± 154.4
Pendulum [-1, 1]	**-235.9** ± 122.7	-237.0 ± 135.5	-236.1 ± 116.3	-312.7 ± 242.5	-252.8 ± 163.0	-236.7 ± 131.6
Pendulum [-2, 2]	-143.2 ± 161.0	-145.5 ± 136.1	-144.5 ± 208.8	-145.2 ± 168.9	-146.2 ± 257.6	**-142.0** ± 133.8
Hopper [-0.25, 0.25]	**343.2** ± 62.6	329.7 ± 59.4	276.9 ± 97.4	237.8 ± 100.0	287.1 ± 76.9	291.2 ± 77.3
Hopper [-0.5, 0.5]	**411.7** ± 115.2	341.7 ± 139.9	342.9 ± 142.1	248.2 ± 113.2	294.5 ± 108.7	316.2 ± 102.0
Hopper [-1, 1]	**459.8** ± 144.9	427.5 ± 151.2	417.2 ± 145.4	245.9 ± 140.7	368.2 ± 139.3	245.9 ± 140.7
Walker2D [-0.25, 0.25]	276.3 ± 118.5	**285.6** ± 97.6	283.7 ± 104.6	219.9 ± 120.8	270.4 ± 104.2	261.0 ± 87.9
Walker2D [-0.5, 0.5]	288.9 ± 118.1	295.6 ± 113.9	304.7 ± 116.1	233.7 ± 99.4	259.0 ± 110.7	**323.4** ± 124.1
HalfCheetah [-0.25, 0.25]	**394.8** ± 43.8	337.4 ± 60.0	339.1 ± 137.9	247.3 ± 96.0	330.7 ± 98.9	314.0 ± 117.6
HalfCheetah [-0.5, 0.5]	718.6 ± 199.9	**736.4** ± 122.8	686.7 ± 224.1	405.1 ± 243.2	456.3 ± 238.5	225.6 ± 179.6
Ant [-0.1, 0.1]	402.3 ± 27.4	**406.2** ± 32.6	378.2 ± 39.7	295.0 ± 44.2	374.0 ± 35.9	346.5 ± 56.9
Ant [-0.25, 0.25]	413.1 ± 60.0	443.1 ± 65.6	451.4 ± 54.8	323.0 ± 60.8	444.2 ± 63.3	**473.9** ± 52.7
Humanoid [-0.1, 0.1]	405.7 ± 112.5	431.9 ± 244.8	397.0 ± 145.7	392.7 ± 169.9	**494.4** ± 182.0	489.6 ± 137.7
Humanoid [-0.25, 0.25]	460.2 ± 143.2	**622.8** ± 158.1	529.8 ± 179.9	374.6 ± 126.5	582.1 ± 176.7	459.6 ± 141.2

Table 1: 95-percentile for the mean and standard deviation of final returns with the best hyper-parameter configuration. The full training curves are given in Figure 1 in Appendix D. CAQL significantly outperforms NAF on most benchmarks, as well as DDPG and TD3 on 10/14 benchmarks. The CAQL-MIP policies are among the top-performing CAQL policies on all the experiments except Ant [-0.25, 0.25] and Humanoid [-0.25, 0.25] and have slightly lower variance.

Environment [Action range]	CAQL-MIP	CAQL-GA	CAQL-CEM	NAF	DDPG	TD3
Pendulum [-0.66, 0.66]	-780.5 ± 345.0	**-766.6** ± 344.2	-784.7 ± 349.3	-775.3 ± 353.4	-855.2 ± 331.2	-886.3 ± 313.6
Pendulum [-1, 1]	-508.1 ± 383.2	-509.7 ± 383.5	**-500.7** ± 382.5	-529.5 ± 377.4	-623.3 ± 395.2	-634.2 ± 381.0
Pendulum [-2, 2]	**-237.3** ± 487.2	-250.6 ± 508.1	-249.7 ± 488.5	-257.4 ± 370.3	-262.0 ± 452.6	-300.0 ± 505.4
Hopper [-0.25, 0.25]	**292.7** ± 93.3	210.8 ± 125.3	196.9 ± 130.1	176.6 ± 109.1	196.4 ± 127.3	162.6 ± 112.5
Hopper [-0.5, 0.5]	**332.2** ± 119.7	222.2 ± 138.5	228.1 ± 135.7	192.8 ± 101.6	218.3 ± 129.6	215.3 ± 111.8
Hopper [-1, 1]	**352.2** ± 141.3	251.5 ± 153.6	242.3 ± 153.8	201.9 ± 126.2	248.0 ± 148.3	201.9 ± 126.2
Walker2D [-0.25, 0.25]	**247.6** ± 109.0	213.5 ± 111.3	206.7 ± 112.9	190.5 ± 117.5	209.9 ± 103.6	202.2 ± 110.1
Walker2D [-0.5, 0.5]	**213.1** ± 120.0	209.5 ± 112.5	209.3 ± 112.5	179.7 ± 100.9	210.8 ± 108.3	196.2 ± 111.6
HalfCheetah [-0.25, 0.25]	**340.9** ± 110.2	234.3 ± 136.5	240.4 ± 143.1	169.7 ± 123.7	228.9 ± 118.1	230.3 ± 134.4
HalfCheetah [-0.5, 0.5]	399.6 ± 274.3	**435.5** ± 273.7	377.5 ± 280.5	271.8 ± 226.9	273.8 ± 199.5	119.0 ± 129.3
Ant [-0.1, 0.1]	319.4 ± 69.3	**327.5** ± 67.5	295.5 ± 71.9	260.2 ± 53.1	298.4 ± 67.6	239.2 ± 59.6
Ant [-0.25, 0.25]	362.3 ± 60.3	388.9 ± 63.9	392.9 ± 67.1	270.4 ± 72.5	381.9 ± 63.3	**398.1** ± 84.4
Humanoid [-0.1, 0.1]	**326.6** ± 93.5	235.3 ± 165.4	227.7 ± 143.1	261.6 ± 154.1	259.0 ± 188.1	283.1 ± 142.1
Humanoid [-0.25, 0.25]	267.0 ± 163.8	**364.3** ± 215.9	309.4 ± 186.3	270.2 ± 124.6	347.3 ± 220.8	303.6 ± 135.0

Table 2: 95-percentile for the mean and standard deviation of final returns over all 320 configurations (32 hyper parameter combinations $\times$ 10 random seeds). The full training curves are given in Figure 2 in Appendix D. CAQL-MIP policies are least sensitive to hyper parameters on 8/14 benchmarks.

We speculate this is due to the fact that the small ReLU NN (32×16) doesn't have enough representation power to accurately model the Q-functions in more complex tasks, and therefore optimizing for the true max-Q value using an inaccurate function approximation indeed cripples the learning.

5 Conclusions and Future Work

In this work, we proposed *Continuous Action Q-learning (CAQL)*, a general framework for handling continuous actions in value-based RL, in which the Q-function is parameterized by a neural network. Any nonlinear optimization methods, ranging from MIP (global optimization) to CEM or GA (zeroth-order or first-order local optimization), that can optimize a generic deep neural network of continuous variables, can be naturally integrated with the framework. We illustrated how the inner maximization of Q-learning can be formulated as mixed-integer programming when the Q-function is parameterized with a ReLU network. We showed that CAQL (with action function learning) is a general Q-learning framework that can be reduced to many existing value-based methods such as QT-Opt and Actor-Expert, when particular optimization methods are selected. Using several benchmark continuous-control problems with varying degrees of action constraints, we showed that the policy learned by CAQL-MIP generally performs better and is more robust to change of hyper parameters than the policies learned by CAQL-GA and CAQL-CEM. We also compared CAQL with several state-of-the-art policy-based RL algorithms and illustrated that CAQL is competitive with policy-based methods and has better performance especially in heavily-constrained environments. Future work includes (i) extending CAQL to the batch learning setting, in which the optimal Q-function is trained using only offline data, (ii) speeding up the MIP computation of the max-Q problem, to make CAQL more scalable, and (iii) applying CAQL to real-world continuous-action RL problems.

References

B. Amos, L. Xu, and Z. Kolter. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 146–155. PMLR, 2017.

R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. P. Vielma. Strong mixed-integer programming formulations for trained neural networks. *arXiv preprint arXiv:1811.01988*, 2019.

C. Boutilier, A. Cohen, A. Hassidim, Y. Mansour, O. Meshi, M. Mladenov, and D. Schuurmans. Planning and learning with stochastic action sets. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 4674–4682. AAAI Press, 2018.

Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *Advances in Neural Information Processing Systems 31*, pp. 8092–8101. Curran Associates, Inc., 2018.

IBM ILOG CPLEX. V12.1: Users manual for CPLEX. 2019. URL `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

G. Dulac-Arnold, D. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

M. Fazel, R. Ge, S. Kakade, and M. Mesbahi. Global convergence of policy gradient methods for the linear quadratic regulator. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1467–1476, Stockholm, Sweden, 2018. PMLR.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1587–1596, Stockholm, Sweden, 2018. PMLR.

C. Gaskett, D. Wettergreen, and A. Zelinsky. Q-learning in continuous state and action spaces. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, pp. 417–428. Springer, 1999.

J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, and X. Ye. Horizon: Facebook's open source applied reinforcement learning platform. *arXiv preprint arXiv:1811.00260*, 2018.

A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, July 2018. URL `http://www.optimization-online.org/DB_HTML/2018/07/6692.html`.

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-learning with model-based acceleration. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp. 2829–2838, New York, NY, USA, 2016. PMLR.

Gurobi. Gurobi optimizer reference manual, 2019. URL `http://www.gurobi.com`.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1861–1870, Stockholm, Sweden, 2018. PMLR.

H. Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems 23*, pp. 2613–2621. Curran Associates, Inc., 2010.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

L. Baird III and A. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical report, Wright Lab Wright-Patterson AFB OH, 1993.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87, pp. 651–673. PMLR, 2018.

A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential Monte Carlo methods. In *Advances in neural information processing systems*, pp. 833–840, 2008.

S. Lim, A. Joseph, L. Le, Y. Pan, and M. White. Actor-Expert: A framework for using action-value methods in continuous action spaces. *arXiv preprint arXiv:1810.09103*, 2018.

L. Lin and T. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical report, Pittsburgh, PA, USA, 1992.

S. Mannor, R. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning*, ICML'03, pp. 512–519. AAAI Press, 2003.

F. Melo and M. Ribeiro. Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pp. 308–322. Springer, 2007.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

G. Neu, A. Jonsson, and V. Gómez. A unified view of entropy-regularized Markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

J. Nocedal and Y. Yuan. Combining trust region and line search techniques. In *Advances in nonlinear programming*, pp. 153–175. Springer, 1998.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225. IEEE, 2006.

M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. *CoRR*, abs/1802.10264, 2018. URL `http://arxiv.org/abs/1802.10264`.

R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology And Computing In Applied Probability*, 1(2):127–190, 1999.

B. Say, G. Wu, Y. Q. Zhou, and S. Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI'17*, pp. 750–756, 2017.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–387–I–395. JMLR.org, 2014.

W. Smart and L. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pp. 903–910, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

R. Sutton, D. McAllester, S. Singh P, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

W. Uther and M. Veloso. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of AAAI '98*, pp. 769–774, 1998.

C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, D. Boning, I. Dhillon, and L. Daniel. Towards fast computation of certified robustness for ReLU networks. *arXiv preprint arXiv:1804.09699*, 2018.

A Hinge Q-learning

Consider an MDP with states X, actions A, transition probability function P, discount factor $\gamma \in [0, 1)$, reward function R, and initial state distribution β. We want to find an optimal Q-function by solving the following optimization problem:

$$\min_{Q} \sum_{x\in X, a\in A} p(x,a)Q(x,a)$$
$$Q(x,a) \geq R(x,a) + \gamma \sum_{x'\in X} P(x'|x,a) \max_{a'\in A} Q(x',a'),\ \forall x \in X, a \in A. \tag{4}$$

The formulation is based on the LP formulation of MDP (see Puterman (2014) for more details). Here the distribution $p(x, a)$ is given by the data-generating distribution of the replay buffer B. (We assume that the replay buffer is large enough such that it consists of experience from almost all state-action pairs.) It is well-known that one can transform the above constrained optimization problem into an unconstrained one by applying a penalty-based approach (to the constraints). For simplicity, here we stick with a single constant penalty parameter $\lambda \geq 0$ (instead of going for a state-action Lagrange multiplier and maximizing that), and a hinge penalty function $(\cdot)_+$. With a given penalty hyper-parameter $\lambda \geq 0$ (that can be separately optimized), we propose finding the optimal Q-function by solving the following optimization problem:

$$\min_{Q} \sum_{x\in X, a\in A} p(x,a)Q(x,a) + \lambda \left(R(x,a) + \gamma \sum_{x'\in X} P(x'|x,a) \max_{a'\in A} Q(x',a') - Q(x,a) \right)_+ . \tag{5}$$

Furthermore, recall that in many off-policy and offline RL algorithms (such as DQN), samples in form of $\{(x_i, a_i, r_i, x'_i)\}_{i=1}^{|B|}$ are independently drawn from the replay buffer, and instead of the optimizing the original objective function, one goes for its unbiased sample average approximation (SAA). However, viewing from the objective function of problem (5), finding an unbiased SAA for this problem might be challenging, due to the non-linearity of hinge penalty function $(\cdot)_+$. Therefore, alternatively we turn to study the following unconstrained optimization problem:

$$\min_{Q} \sum_{x\in X, a\in A} p(x,a)Q(x,a) + \lambda \sum_{x'\in X} P(x'|x,a) \left(R(x,a) + \gamma \max_{a'\in A} Q(x',a') - Q(x,a) \right)_+ . \tag{6}$$

Using the Jensen's inequality for convex functions, one can see that the objective function in (6) is an upper-bound of that in (5). Equality of the Jensen's inequality will hold in the case when transition function is deterministic. (This is similar to the argument of PCL algorithm.) Using Jensen's inequality one justifies that optimization problem (6) is indeed an eligible upper-bound optimization to problem (5).

Recall that $p(x, a)$ is the data-generation distribution of the replay buffer B. The unbiased SAA of problem (6) is therefore given by

$$\min_{Q} \frac{1}{N} \sum_{s=1}^{N} Q(x_i, a_i) + \lambda \left(r_i + \gamma \max_{a'\in A} Q(x'_i, a') - Q(x_i, a_i) \right)_+ , \tag{7}$$

where $\{(x_i, a_i, r_i, x'_i)\}_{s=1}^{N}$ are the N samples drawn independently from the replay buffer. In the following, we will find the optimal Q function by solving this SAA problem. In general when the state and action spaces are large/uncountable, instead of solving the Q-function exactly (as in the tabular case), we turn to approximate the Q-function with its parametrized form Q_θ, and optimize the set of real weights θ (instead of Q) in problem (7).

B Continuous Action Q-learning Algorithm

Algorithm 1 Continuous Action Q-learning

Initialize the Q-function parameters θ, target Q-function parameters θ^{target}, and action function parameters w
Initialize replay buffer R
for $i \leftarrow 1, \ldots, M$ **do**
  for $t \leftarrow 1, \ldots, T$ **do**
    Select action $a_t = clip(\pi_w(x_t) + \mathcal{N}(0, \sigma), l, u)$
    Execute action a_t and observe reward r_t and new state x_{t+1}
    Store transition (x_t, a_t, r_t, x_{t+1}) in R
  for $s \leftarrow 1, \ldots, K$ **do** ▷ K=20 by default
    Sample a random minibatch of N transitions (x_i, a_i, r_i, x'_i) from R
    Compute the optimal action for x'_i in the minibatch:

$$a'_i = \arg\max_{a'} Q_\theta(x'_i, a')$$

    Compute TD targets for the minibatch:

$$q_i = r_i + \gamma Q_{\theta^{\text{target}}}(x'_i, a'_i)$$

    Update the Q-function parameters:

$$\theta \leftarrow \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \left(Q_\theta(x_i, a_i) - q_i\right)^2$$

    Update the action function parameters:

$$w \leftarrow \arg\min_w \frac{1}{N} \sum_{i=1}^{N} (Q_\theta(x'_i, a'_i) - Q_\theta(x'_i, \pi_w(x'_i)))^2$$

    Update the target Q-function parameters:

$$\theta^{\text{target}} \leftarrow \tau\theta + (1 - \tau)\theta^{target}$$

  Decay the Gaussian noise:

$$\sigma \leftarrow \lambda\sigma, \lambda \in [0, 1]$$

C Experimental Details

We use a two hidden layer neural network with ReLU activation (32 units in the first layer and 16 units in the second layer) for both the Q-function and the action function. The input layer for the Q-function is a concatenated vector of state representation and action variables. The Q-function has a single output unit (without ReLU). The input layer for the action function is only the state representation. The output layer for the action function has d units (without ReLU), where d is the action dimension of a benchmark environment. We use SCIP 6.0.0 (Gleixner et al., 2018) for the MIP solver. A time limit of 60 seconds and a optimality gap limit of 10^{-4} are used for all experiments.

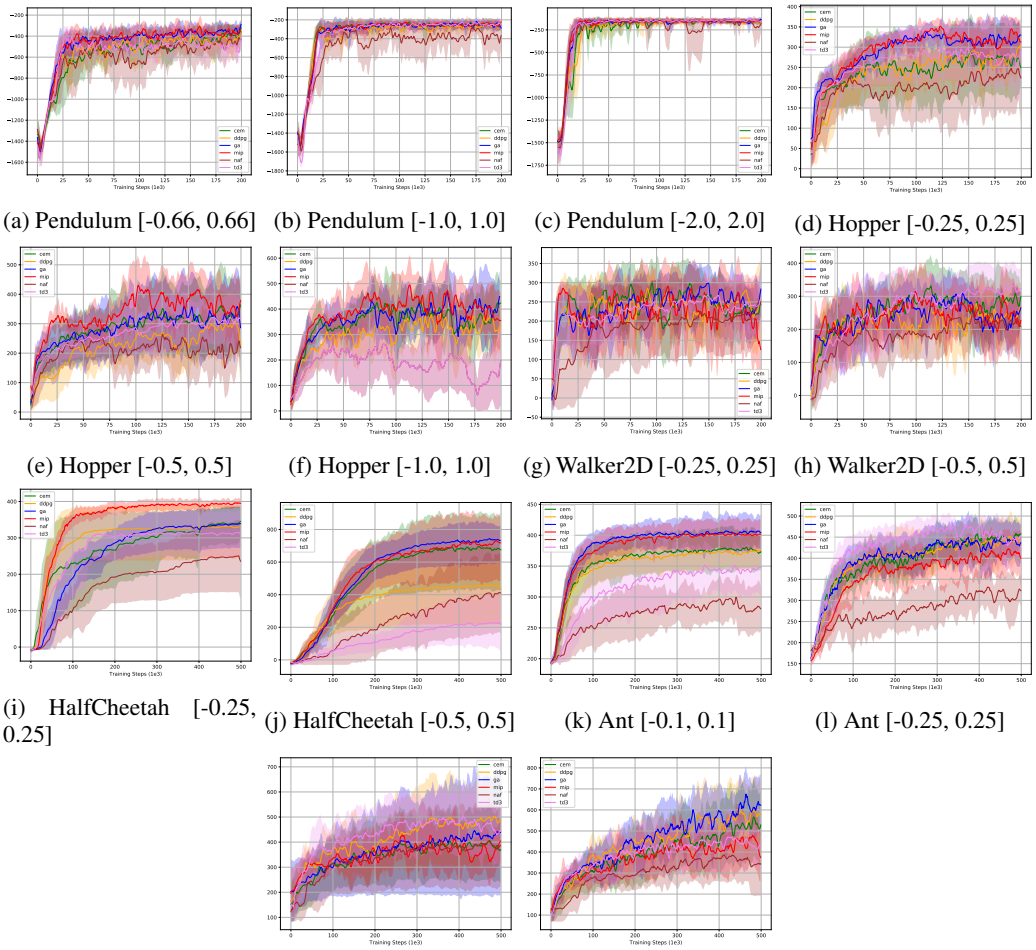
Environment	State dimension	Action dimension	Action ranges
Pendulum	3	1	**[-2, 2]**, [-1, 1], [-0.66, 0.66]
Hopper	11	3	**[-1, 1]**, [-0.5, 0.5], [-0.25, 0.25]
Walker2D	17	6	[-0.5, 0.5], [-0.25, 0.25]
HalfCheetah	17	6	[-0.5, 0.5], [-0.25, 0.25]
Ant	111	8	[-0.25, 0.25], [-0.1, 0.1]
Humanoid	376	17	[-0.25, 0.25], [-0.1, 0.1]

Table 3: Benchmark Environments. Various action bounds are tested from the default one to smaller ones. The action range in bold is the default one. For high-dimensional environments such as Walker2D, HalfCheetah, Ant, and Humanoid, we use action ranges smaller than the default (i.e., [-1, 1]) due to the long computation time for MIP. A smaller action bound results in a MIP that solves faster.

Hyper Parameter	Value(s)
Discount factor	0.99
Exploration policy	$\mathcal{N}(0, \sigma = 1)$
Exploration noise (σ) decay	0.9995, 0.9999
Exploration noise (σ) minimum	0.025
Soft target update rate (τ)	0.001
Replay memory size	10^5
Mini-batch size	64
Q-function learning rates	0.001, 0.0005, 0.0002, 0.0001
Action function learning rates	0.001, 0.0005, 0.0002, 0.0001
Neural network optimizer	Adam

Table 4: All methods (CAQL(+ MIP, GA, CEM), NAF, DDPG, TD3) use the same hyper parameters. We sweep over the Q-function learning rates, action function learning rates, and exploration noise decays.

D Additional Experimental Results



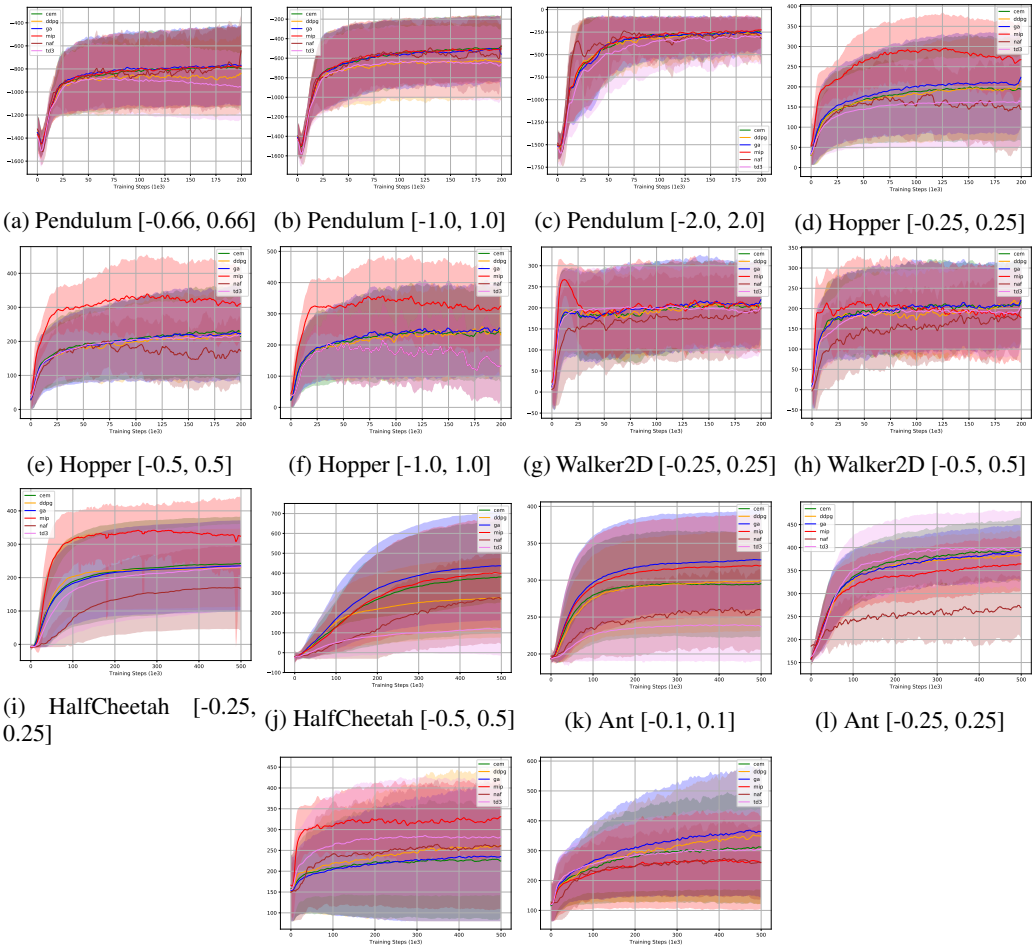
(a) Pendulum [-0.66, 0.66] (b) Pendulum [-1.0, 1.0] (c) Pendulum [-2.0, 2.0] (d) Hopper [-0.25, 0.25]

(e) Hopper [-0.5, 0.5] (f) Hopper [-1.0, 1.0] (g) Walker2D [-0.25, 0.25] (h) Walker2D [-0.5, 0.5]

(i) HalfCheetah [-0.25, 0.25] (j) HalfCheetah [-0.5, 0.5] (k) Ant [-0.1, 0.1] (l) Ant [-0.25, 0.25]

(m) Humanoid [-0.1, 0.1] (n) Humanoid [-0.25,0.25]

Figure 1: Mean cumulative reward over 10 random seeds. Shaded area is $\pm$ standard deviation. Data points are average over a sliding window of size 3. The length of an episode is limited to 200 steps.



(a) Pendulum [-0.66, 0.66] (b) Pendulum [-1.0, 1.0] (c) Pendulum [-2.0, 2.0] (d) Hopper [-0.25, 0.25]

(e) Hopper [-0.5, 0.5] (f) Hopper [-1.0, 1.0] (g) Walker2D [-0.25, 0.25] (h) Walker2D [-0.5, 0.5]

(i) HalfCheetah [-0.25, 0.25] (j) HalfCheetah [-0.5, 0.5] (k) Ant [-0.1, 0.1] (l) Ant [-0.25, 0.25]

(m) Humanoid [-0.1, 0.1] (n) Humanoid [-0.25,0.25]

Figure 2: Mean cumulative reward over all 320 configurations (32 hyper parameter combinations $\times$ 10 random seeds). Shaded area is $\pm$ standard deviation. Data points are average over a sliding window of size 3. The length of an episode is limited to 200 steps.