Q-Learning in enormous action spaces via amortized approximate maximization

Tom Van de Wiele,[*] David Warde-Farley, Andriy Mnih & Volodymyr Mnih
DeepMind
London, United Kingdom
tvdwiele@gmail.com, {dwf,amnih,vmnih}@google.com

Abstract

Applying Q-learning to high-dimensional or continuous action spaces can be difficult due to the required maximization over the set of possible actions. Motivated by techniques from amortized inference, we replace the expensive maximization over all actions with a maximization over a small subset of possible actions sampled from a learned proposal distribution. The resulting approach, which we dub Amortized Q-learning (AQL), is able to handle discrete, continuous, or hybrid action spaces while maintaining the benefits of Q-learning. Our experiments on continuous control tasks with up to 21 dimensional actions show that AQL outperforms D3PG (Barth-Maron et al., 2018) and QT-Opt (Kalashnikov et al., 2018). Experiments on structured discrete action spaces demonstrate that AQL can efficiently learn good policies in spaces with thousands of discrete actions.

1 Introduction

In the recent resurgence of interest in combining reinforcement learning with neural network function approximators, Q-learning (Watkins & Dayan, 1992) and its many neural variants (Mnih et al., 2015; Bellemare et al., 2017) have remained competitive with the state-of-the-art methods (Horgan et al., 2018; Kapturowski et al., 2019). The simplicity of Q-learning makes it relatively straightforward to implement, even in combination with neural networks (Mnih et al., 2015). Because Q-learning is an off-policy reinforcement learning algorithm, it is trivial to combine with techniques like experience replay (Lin, 1993) for improved data efficiency or to implement in a distributed training setting where experience is generated using slightly stale network parameters, without requiring importance sampling-based off-policy corrections used by stochastic actor-critic methods (Espeholt et al., 2018). Q-learning can also be easily and robustly combined with exotic exploration strategies (Ostrovski et al., 2017; Mnih et al., 2016; Horgan et al., 2018) as well as data augmentation through post-hoc modification (Kaelbling, 1993; Andrychowicz et al., 2017).

One limitation of Q-learning is the requirement to maximize over the set of possible actions, limiting its applicability in environments with continuous or high-cardinality discrete action spaces. In the case of Q-learning with neural network function approximation, the common approach of computing Q values for all actions in a single forward pass becomes infeasible, requiring instead an architecture which accepts as inputs both the state and the action, producing a scalar Q estimate as output. Other types of methods, such stochastic actor-critic or policy gradient approaches can naturally handle discrete, continuous, or even hybrid action spaces (Williams, 1992; Schulman et al., 2015; Mnih et al., 2016) because they do not perform a maximization over actions and only require the ability to efficiently sample actions from an appropriately parameterized stochastic policy. Furthermore, the structure of an action space often suggests a stochastic policy parametrization which admits efficient sampling even when the number of distinct actions is enormous. For example, a K-level discretization of a continuous action space with D degrees of freedom (throughout this work, we will refer to this as a *D-dimensional* action space) will have K^D distinct actions, but a policy can be designed to represent these actions as a product of D discrete distributions each with arity K, allowing for sampling in $\mathcal{O}(KD)$ rather than $\mathcal{O}(K^D)$ time. The same principle cannot be directly applied to Q-learning, where identification of the maximal Q value would require an exhaustive

[*] Work done while at DeepMind. Address correspondence to {dwf,vmnih}@google.com

enumeration of all K^D actions. While a number of approaches for dealing with the intractable maximization over actions in Q-learning have been proposed in order to make it applicable to richer action spaces, these approaches are usually specific to a particular form of action space (Hafner, 2009; Pazis & Lagoudakis, 2009; Yoshida, 2015).

Here, we show that instead of performing an exact maximization over the set of actions at each time step, it can be preferable to *learn* to search for the best action, thus amortizing the action selection cost over the course of training (Hafner, 2009; Lillicrap et al., 2016). We treat the search for the best action as another learning problem and replace the exact maximization over all actions with a maximization over a set of actions sampled from a learned proposal distribution. We show that an effective proposal can be learned by training a neural network to predict the best known action found by a stochastic search procedure. This approach, which we dub Amortized Q-learning (AQL), is able to naturally handle high-dimensional discrete, continuous, or even hybrid action spaces (consisting of both discrete and continuous degrees of freedom) because, like stochastic actor-critic methods, AQL only requires the ability to sample actions from an appropriately parameterized proposal distribution.

We evaluate the effectiveness of Amortized Q-learning on both continuous actions spaces and large, but structured, discrete action spaces. On continuous control tasks from the DeepMind Control Suite, Amortized Q-learning outperforms D3PG and QT-Opt, two strong continuous control methods. On foraging tasks from the DeepMind Lab 3D first-person environment, Amortized Q-learning is able to learn effective policies using an action space with over 3500 actions. AQL thus bridges the gap between Q-learning and stochastic actor-critic methods in their flexibility towards the action space, removing the need for Q-learning methods tailored to specific action spaces.

2 Background

We consider discrete time reinforcement learning problems with large action spaces. At time step t the agent observes a state s_t and produces an action $\mathbf{a}_t$. The agent then receives a reward r_t and transitions to the next state s_{t+1} according to the environment transition distribution defined as $p_\mathcal{E} \triangleq p(r_t, s_{t+1}|s_t, \mathbf{a}_t)$. The aim of the agent is to maximize the expected future discounted return $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$ where $\gamma \in [0,1]$ is a discount factor. While in the standard Markov Decision Process formulation, the actions $\mathbf{a}_t$ come from a finite set of possible actions $\mathcal{A}$, we consider a more general case. Specifically, we assume that the action space is combinatorially structured, i.e. defined as a Cartesian product of sub-action spaces. Formally we assume $\mathcal{A} = \mathcal{A}_1 \times \ldots \mathcal{A}_D$ where each $\mathcal{A}_i$ is either a finite set or $\mathcal{A}_i = [a,b] \subset \mathbb{R}$. We therefore consider $\mathbf{a}_t$ to be a D-dimensional vector and each component of $\mathbf{a}_t$ is referred to as a sub-action.

Given a policy π that maps states to distributions over actions, the action-value function for policy π is defined as $Q_\pi(s,\mathbf{a}) = \mathbb{E}_{\pi,p_\mathcal{E}}[R_t|s_t = s, \mathbf{a}_t = \mathbf{a}]$. The optimal action-value function defined as $Q^*(s,\mathbf{a}) = \max_\pi Q_\pi(s,\mathbf{a})$ gives the expected return for taking action $\mathbf{a}$ in state s and acting optimally thereafter. If Q^* is known, an optimal deterministic policy π^* can be obtained by acting greedily with respect to Q^*, i.e. taking $\pi^*(s) = \arg\max_\mathbf{a} Q^*(s,\mathbf{a})$. An important property of Q^* is that it can be decomposed recursively according to the Bellman equation

$$Q^*(s,\mathbf{a}) = \mathbb{E}_{p_\mathcal{E}}\left[r_t + \gamma \max_{\mathbf{a}'} Q^*(s_{t+1},\mathbf{a}')|s_t = s, \mathbf{a}_t = \mathbf{a}\right] \tag{1}$$

In value-based reinforcement learning methods the aim is to learn the optimal value function Q^* by starting with a parametric estimate $Q(s,\mathbf{a};\theta)$ and iteratively improving the parameters θ based on experience sampled from the environment.

The Q-learning algorithm (Watkins & Dayan, 1992) is one such value-based method, which uses an iterative update based on the recursive relationship in the Bellman equation to learn $Q(s,\mathbf{a};\theta)$. Specifically, Q-learning can be formalized as optimizing the loss

$$L(\theta) = \mathbb{E}_{\pi,p_\mathcal{E}}\left[\left(r_t + \gamma \max_{\mathbf{a}} Q(s_{t+1},\mathbf{a};\bar{\theta}) - Q(s_t,\mathbf{a}_t;\theta)\right)^2\right], \tag{2}$$

where $\bar{\theta} = \theta$ but $\bar{\theta}$ is treated as a constant for the purposes of optimization, i.e. no gradients flow through it.

3 RELATED WORK

Deterministic policy gradient algorithms (Hafner, 2009; Silver et al., 2014; Lillicrap et al., 2016) learn a deterministic policy that maps states to continuous actions by following the gradient of an action-value function critic with respect to the action $\mathbf{a}$. Silver et al. (2014) justified this approach by proving the deterministic policy gradient theorem, which shows how the gradient of an action-value function Q_π with respect to the action $\mathbf{a}$ at state s can be used to improve the policy π at state s. As others have noted (Haarnoja et al., 2017), these methods can be interpreted as Q-learning, with the deterministic policy serving as an approximate maximizer over the action space, partially explaining why such methods work well with off-policy data. Taking this view makes deterministic policy gradient methods similar to our proposed method. However, AQL learns an approximate maximizer using explicit search and supervised learning rather than using the gradient of the critic. This lack of reliance on gradients with respect to actions renders AQL agnostic to the type of the action space, be it discrete, continuous or a combination of the two, while deterministic policy gradients are inapplicable to discrete action spaces.

Gu et al. (2016) address the intractability of Q-maximization in the continuous setting by restricting the form of Q to be a state-dependent quadratic function of the continuous actions, rendering the maximization trivial. While considerably simpler than DDPG, the practical consequences of this representational restriction in the continuous case are unclear, and the technique is inapplicable to discrete or hybrid action spaces.

Metz et al. (2018) apply Q-learning to multi-dimensional continuous action spaces by discretizing each continuous sub-action. Their Sequential DQN approach avoids maximizing over a number of actions that is exponential in the number of action sub-actions D by considering an extended MDP in which the agent chooses its sub-actions sequentially. Sequential DQN learns Q functions for the original MDP as well as the D extended MDPs, using the latter to perform Q-learning backups for the former across environment transitions. Tavakoli et al. (2017) model Q-values for all sub-actions using a shared state value and a sub-action-specific advantage parametrization. Their best working parametrization subtracts the mean advantage for each action advantage dimension. The target Q-value is shared for all action dimensions and computed using the observed rewards and a bootstrapped value equal to the mean argmax action of the target network. Notably, Metz et al. (2018) conditions the lower-level Q function on sub-actions already taken as part of the current macro-action; we experiment with a similar strategy for our learned proposal distribution.

Another alternative to performing an exact maximization over actions is to replace this maximization with a fixed stochastic search procedure. Kalashnikov et al. (2018) and Quillen et al. (2018) used the cross-entropy method to perform an approximate maximization over actions in order to apply Q-learning to robotic grasping tasks. This approach has been shown to work very well for tasks with up to 6-dimensional action spaces, but it is unclear if it can scale to higher-dimensional action spaces. Notably, concurrent work[1] of Lim et al. (2018) develops an *Actor-Expert* framework wherein an approximate Q function is trained on continuous actions and a stochastic policy is updated with a state-conditional variant of the cross-entropy method. Our work demonstrates that a simpler approach, rendered efficient by parallel computation of Q values, can scale to even larger continuous action spaces, as well as high-dimensional discrete and hybrid action spaces.

On the subject of replacing policies with proposal distributions, concurrent work of Hunt et al. (2019) learns proposal distributions in the stochastic case. Their proposal distribution consists of a mixture of truncated normal distributions that is learned by minimizing the forward KL divergence with the Boltzmann policy. This method is focused on sampling in continuous action spaces during transfer, and does not deal with discrete or hybrid action spaces.

Finally, we note also the concurrent work of Wiehe et al. (2018) applied the idea of learning an approximate maximizer over actions using search and supervised learning to continuous control. Our work shows that this idea is more generally applicable and our experiments validate it on more challenging tasks.

[1] Versions of both Lim et al. (2018) and the present manuscript appeared concurrently at the NeurIPS 2018 Deep Reinforcement Learning Workshop.

Algorithm 1: AQL

```
Input : Proposal network parameters $\theta^\mu$, $Q$ network parameters $\theta^Q$, number of actions to draw
        from the proposal $N$, number of actions to draw uniformly $M$, unroll length $T$,
        exploration probability $\epsilon$
repeat
    for $t \leftarrow 1 \dots T$ do
        Observe state $s_t$
        $A_U := \{\mathbf{a}_j^U\}_{j=1}^M,\ \mathbf{a}_j^U \sim \text{Uniform}(\mathcal{A}_1 \times \dots \times \mathcal{A}_D)$
        $A_\mu := \{\mathbf{a}_i^\mu\}_{i=1}^N,\ \mathbf{a}_i^\mu \sim \mu(\mathbf{a}|s_t;\theta^\mu)$
        $\mathbf{a}_t^* := \arg\max_{\mathbf{a} \in A_U \cup A_\mu} Q(s_t, \mathbf{a})$
        with probability $\epsilon$,
            $\mathbf{a}_t := \mathbf{a}_1^U$ // Select $\mathbf{a}_t$ uniformly at random
        otherwise
            $\mathbf{a}_t := \mathbf{a}_t^*$
        $r_t, s_{t+1} \sim p_E(r_t, s_{t+1}|s_t, \mathbf{a}_t)$ // Take action $\mathbf{a}_t$, receive reward $r_t$
    end
    Update $\theta^Q$ using $s_{1:T}$, $\mathbf{a}_{1:T}$, $r_{1:T}$ by descending the gradient of (4).
    Update $\theta^\mu$ using $s_{1:T}$ and $\mathbf{a}_{1:T}^*$ by descending the gradient of (3).
until termination
```

4 Amortized Q-Learning

Amortized inference (Dayan et al., 1995) has revolutionized variational inference by eliminating the need for a costly maximization with respect to the parameters of the approximate variational posterior, making training latent variable models much more efficient. Key to this approach is a form of *amortization* in which an expensive procedure such as iterative optimization over the variational parameters is replaced with the more efficient forward pass in a parametric model, such as a neural network. By training the model by optimizing the same objective as the original procedure, we obtain an efficient approximation to it that can be applied to new instances of the problem. We propose to leverage the same principle in order to amortize the maximization of $Q(s, \mathbf{a})$ with respect to the vector of sub-actions $\mathbf{a}$. Specifically, in addition to a neural network representing the Q function (which takes the state s and vector of sub-actions $\mathbf{a}$ as inputs) we train an additional neural network to predict, from the state s, the sufficient statistics of a proposal distribution μ over possible actions. We then replace exact maximization of $Q(s, \mathbf{a})$ over actions $\mathbf{a}$, which is required both for acting greedily and performing Q-learning updates, with maximization over a set of samples from the proposal μ.

As long as the proposal distribution assigns non-zero probability to all actions, this method will approach Q-learning with exact maximization for discrete action spaces as the number of samples from the proposal increases[2]. Since the computational cost of the approach will grow linearly with the number of samples taken from the proposal, for practical reasons we are interested in learning a proposal from which we need to draw the fewest possible samples.

In this paper, we consider a simple approach to learning a proposal distribution which, at a high level, works as follows. To update the proposal for state s we first perform an approximate stochastic search for the action with the highest Q-value at state s. We then update the proposal to predict the best action found by the search procedure, i.e. the one with the highest Q-value, using supervised learning to make the action found by the search procedure more probable under the proposal distribution. Because the proposal is meant to track approximately maximal actions for a given state as prescribed by a parametric Q function which is itself continually changing, we add a regulariza-

[2] To see why this is true, let p_{best} be the probability assigned to the action with the highest Q-value by the proposal. Then after drawing n samples from the proposal, the probability that the best action has been sampled at least once is $1 - (1 - p_{best})^n$ which tends to 1 as n tends to infinity.

tion term to the proposal objective in order to prevent the proposal from becoming deterministic and potentially becoming stuck in a bad local optimum.

More formally, we update the proposal distribution μ for a state s, by drawing a set of M samples from the uniform distribution over all actions and a set of N samples from the current proposal for state s. Denoting the action with the highest Q-value from the union of these two sets as $\mathbf{a}^*(s)$, the regularized loss for training the proposal distribution is given by

$$\mathcal{L}(\theta^\mu; s) = -\log \mu(\mathbf{a}^*(s)|s; \theta^\mu) - \lambda H(\mu(\mathbf{a}|s; \theta^\mu)). \tag{3}$$

The first term in the objective is the negative log-likelihood for the proposal with $\mathbf{a}^*$ as the target. Minimizing it makes the sample with the highest Q value more likely under the proposal. The second term is the negative entropy of the proposal distribution, which encourages uncertainty in μ throughout training and prevents it from collapsing to a deterministic distribution.

Finally, we can define the AQL loss for updating the action-value function as

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{\pi, p_{\mathcal{E}}} \left[\left(r_t + \gamma Q(s_{t+1}, \boxed{\mathbf{a}^*(s_{t+1})}) - Q(s_t, \mathbf{a}_t; \theta^Q) \right)^2 \right], \tag{4}$$

which is identical to the Q-learning loss but with maximization over all actions replaced by the stochastic search procedure based on the proposal distribution. Pseudocode for the complete Amortized Q-learning algorithm is shown in Algorithm 2. While the algorithm considers the action-value and proposal parameters separate, we consider the possibility of sharing some of the parameters between the two following standard practice. The parameter sharing is made clear in the network architecture diagram in Figure 1, which details the architecture used for all experiments.

The AQL algorithm can be applied to continuous, discrete, and hybrid discrete/continuous action spaces simply by changing the form of the proposal distribution μ. We use autoregressive proposals of the form

$$\mu(\mathbf{a}|s; \theta^\mu) = \textstyle\prod_{d=1}^{D} \mu_d(a_d|s, \mathbf{a}_{<d}; \theta^\mu),$$

in order to incorporate the dependencies among sub-actions. For discrete sub-actions we parameterized the proposal μ_d using a softmax. We experimented with discretized and continuous proposal distributions for continuous sub-actions. Proposal distributions for continuous sub-actions are either parameterized using a softmax over a set of discrete choices representing uniformly-spaced values from the continuous sub-action's range or using a Gaussian distribution with fixed variance. A detailed description of the architecture is included in the Appendix.

5 Experiments

We performed experiments on two families of tasks: the set of DeepMind Control Suite (Tassa et al., 2018) tasks with constrained actions (16 distinct domains, with a total of 39 tasks spread between them) and two tasks in the DeepMind Lab (Beattie et al., 2016) 3D first-person environment. Descriptions of the architectures and hyperparameter settings are available in the Appendix.

5.1 Control Suite

The DeepMind Control Suite is a collection of continuous control tasks built on the MuJoCo physics engine (Todorov et al., 2012). The Control Suite is considered a good benchmark as it spans a wide range of tasks of varying action complexity. In the simplest tasks, the agent controls only a single actuator while the `humanoid` tasks require selection of at least 21 sub-actions at every time step. As is common, we employed the underlying state variables rather than visual representations of the state as observations, as the focus of our inquiry is the complexity of the action space rather than the observation space. We explore a domain where the observations are pixel renderings in Section 5.2.

We compared AQL to several baseline continuous control methods. First, we considered an ablation which replaces the learned proposal distribution with a fixed uniform proposal distribution. We dub this method **Uniform Q-learning**. We also considered two strong continuous control methods: D3PG (Barth-Maron et al., 2018), a distributed version of DDPG (Lillicrap et al., 2016), and QT-Opt (Kalashnikov et al., 2018) which corresponds to Q-learning where action maximization is performed using the cross-entropy method (CEM) (Kalashnikov et al., 2018; Rubinstein & Kroese,

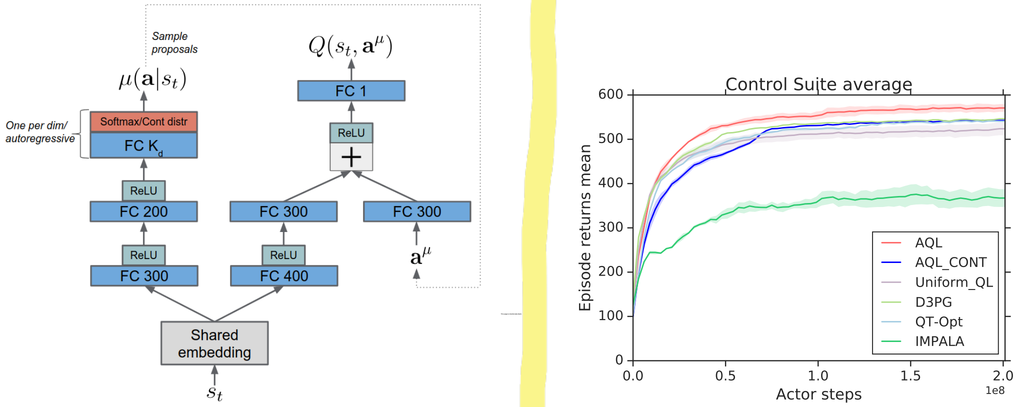

Figure 1: **Left:** The AQL architecture used in the experiments. The shared state embedding network is the identity function for the Control Suite experiments (i.e. the proposal network and Q network each operate directly on the observations and share no parameters). For DeepMind Lab experiments, where the input consists of pixel observations, the shared state embedding consists of 3 layers of ResNet blocks followed by a fully connected layer with 256 units and a recurrent LSTM core with 256 cells as in Espeholt et al. (2018). The left head represents the proposal distribution. There is a proposal output layer of dimension K_d for each sub-action d. K_d represents the number of choices for dimension d for discrete or discretized continuous actions and the number of distribution parameters for continuous actions. Samples from the proposal distribution are embedded and concatenated with an embedding of the state and used to compute $Q(s_t, \mathbf{a}^\mu)$. **Right:** Learning curves of the mean return across all tasks in the Control Suite. The error bars represent the standard error of the mean episode return.

2004). CEM is a derivative-free iterative optimization algorithm that samples K values at each iteration, fits a Gaussian distribution to the best $L < K$ of these samples, and then samples the next batch of K from that distribution. The iterative process is initialized with uniform samples from the action space. In our implementation, we used the same number of initial samples as the number of proposal actions $K = N = 100$ and $L = 10$, performed three iterations and fit independent Gaussian distributions for each sub-action to the L actions with the highest corresponding Q-values. Following previous work (Kalashnikov et al., 2018; Hafner et al., 2018), L is chosen to be an order of magnitude smaller than K. The final baseline method consisted of the IMPALA agent (Espeholt et al., 2018), an importance weighted advantage actor-critic method that is inspired by A3C (Mnih et al., 2016).

Our AQL implementation used an autoregressive proposal distribution which models sub-actions sequentially using the ordering in which they appear in the Control Suite task specifications. We considered both discretized categorical and Gaussian action distributions, the latter with fixed variance $\sigma^2 = 0.25$. The logits (in the discretized case) or distribution means (continuous case) of the proposal distribution for sub-action i are computed by a linear transformation of a concatenation of features derived from the state and one-hot encodings of all previously sampled sub-actions $1, 2, \ldots, i-1$. Note that entire composite actions sampled from the proposal distribution remain independent and identically distributed.

For the discretized AQL method and the Uniform Q-learning method, actions were discretized for each degree of freedom to one of five sub-actions in $\{-1, -0.5, 0, 0.5, 1\}$. This yields a total of 5^D distinct actions. All methods except D3PG used $N = 100$ proposal samples and $M = 400$ uniform search samples as described in Algorithm 2. We considered more uniform search samples since they are less expensive to compute, as they do not require the relatively expensive autoregressive sampling procedure. The Q-value evaluations were implemented as a single batched operation for each actor step, which rendered evaluation highly efficient. Following the literature, our D3PG experiments made use of an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) for exploration with $\theta = 0.15$ and $\sigma = 0.2$. We employed the Adam optimizer (Kingma & Ba, 2014) to train all models. In order fairly compare across methods, all architectural details that are not specific to the

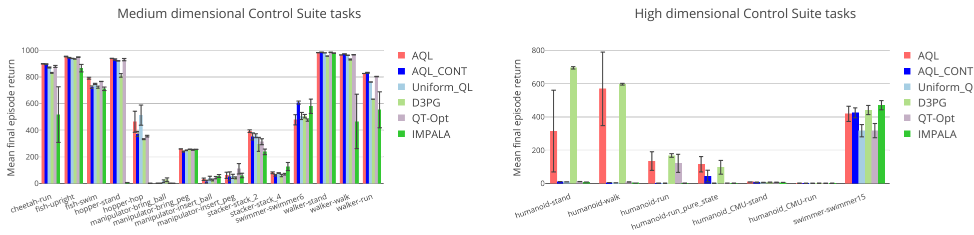

Figure 2: Mean final performance, averaged over 3 seeds, for the medium- and high-dimensional Control Suite tasks. The error bars represent the standard error of the mean episode return.

method under consideration were held fixed across conditions. Further details are included in the Appendix.

Figure 1 depicts the learning curves for 200 million actor steps averaged over all tasks (task specific learning curves are reported in the Appendix), which shows that discretized AQL performs best on average. Returns at a given point in training were obtained by allowing the agent to act in the environment for 1000 time steps and summing the rewards. Rewards range between 0 and 1 for all tasks in the Control Suite, which makes the average rewards comparable between tasks. While not shown in Figure 1 we note that all methods also substantially outperform the A3C results reported in (Tassa et al., 2018) except for the IMPALA results which are only marginally better.

Closer inspection revealed that the difference between the methods in Figure 1 was mostly explained by the performance on the medium- and high-dimensional tasks. Exploration in high-dimensional action spaces is typically a harder problem, and the same appears to hold for searching for the action with the maximum Q-value. Analysis of tasks with high-dimensional action spaces revealed that learning a discretized proposal distribution (AQL) or a deterministic policy (D3PG) clearly outperformed Uniform Q-learning and QT-Opt, both of which use fixed stochastic search procedures for action selection. IMPALA performed worse than the other methods on most Control Suite tasks.

The continuous implementation of AQL performed similarly on the low- and medium-dimensional tasks as the discretized AQL variant but failed to learn on the high-dimensional tasks. This may be related to the observation that exploration is often easier when the action space is discretized (Metz et al., 2018; Peng & van de Panne, 2017). Figure 2 shows the average mean final performance for the medium- and high-dimensional tasks. Results on the low-dimensional Control Suite tasks are available in the Appendix and show less drastic differences between the methods. These experimental results support the hypothesis that learning a proposal distribution is beneficial for more complex or higher-dimensional action spaces.

Overall, these results demonstrate that AQL can scale up to problems with a relatively high-dimensional action space. The 21-dimensional `humanoid` tasks are of particular interest: although our discretization scheme leads to a total of 5^{21} possible discrete actions on these tasks, AQL was able to learn policies competitive with D3PG.

Uniform Q-learning and QT-Opt, both of which correspond to Q-learning with a fixed procedure for maximizing over actions, largely failed to learn on the `humanoid tasks`, demonstrating the advantage of learning a state-dependent maximization procedure in high-dimensional action spaces.

5.2 DeepMind Lab

DeepMind Lab (Beattie et al., 2016) is a platform for 3D first person reinforcement learning environments. We evaluated AQL on DeepMind Lab because it offers a combination of a rich observation space (we consider 84 × 84 RGB observations) and a complex, structured action space.

The full action space consists of 7 discrete sub-actions which can be selected independently. The first two sub-actions specify the angular velocity of the viewport rotation (left/right and up/down). Each of these represents an integer between -512 and 512. The next two sub-actions represent sets of three mutually exclusive movement-related options: strafe left/right/no-op and forward/backward/no-op. The last three sub-actions are binary: fire/no-op, jump/no-op and crouch/no-op. We considered two action sets: a curated set with 11 actions based on the minimal subset that allows good performance on all tasks, and a second, larger action set with 3528 actions consisting of the Cartesian product of 7 rotation actions for each of the two rotational axes and the 5 remaining ternary and binary sub-actions ($3528 = 7^2 \cdot 3^2 \cdot 2^3$).

We compared three methods on each of the two action sets: our distributed implementation of AQL with $N = 100$ proposal samples and $M = 500$ uniform search samples, an ablation that performs exact Q-learning (i.e. evaluates the Q-values for all possible discrete actions at every time step) and the IMPALA method. As with our AQL implementation, the Q-learning ablation agent was distributed, replay-based, and used a recurrent Q-function trained with $Q(\lambda)$ making it a strong baseline similar to R2D2 (Kapturowski et al., 2019). Other details such as network architecture and optimizer choice were held fixed across conditions. The AQL implementation for the large action set incorporated the 7-dimensional structure of the discrete action space and used the same style of autoregressive proposal as the Control Suite experiments. Every action was repeated 4 times and each seed was run for 200 million environment steps (50 million actor steps given the repeated actions). The precise network architecture is described in the Appendix.

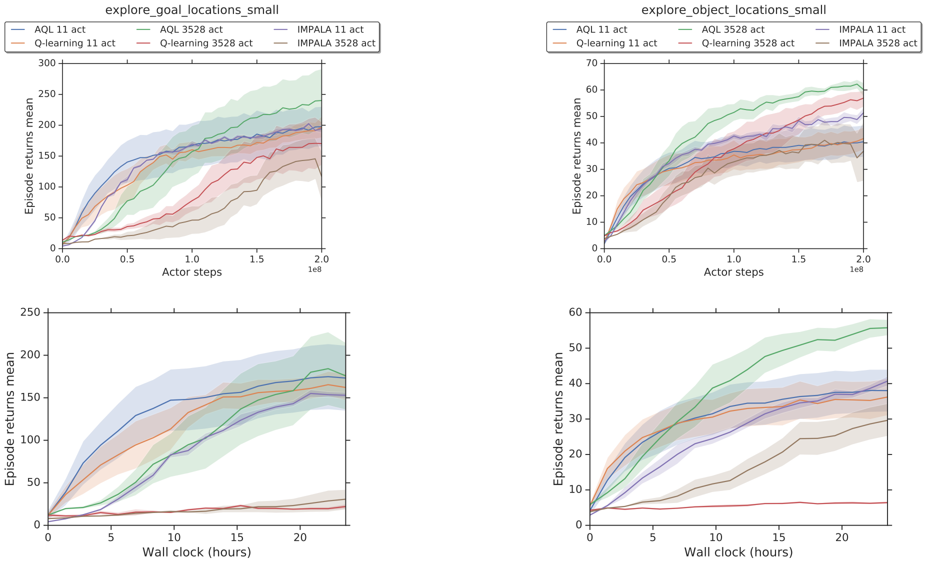

Figure 3: Learning curves on two DeepMind Lab exploration tasks. The error bars represent the standard error of the mean episode return over 5 seeds. The top row shows the mean episode return versus the number of actor steps and the bottom row represent the mean episode return versus time for the first 24 hours of the experiment.

Figure 3 shows the results for two exploration tasks from DeepMind Lab. In both tasks the agent was rewarded for navigating to the object(s) in a maze. The `explore_goal` task only contains a single object with goal location(s), level layout and theme randomized per episode. As expected, there was no significant performance difference between AQL and exact Q-learning (i.e. computing Q-values for all 11 actions) when using the curated action set. IMPALA also performs comparably with the curated action set. When training with the large action set, exploration is harder, which explains why the initial performance was worse for all methods. AQL achieved the best final score on both tasks, eventually outperforming IMPALA, exact Q-learning with the curated action set and exact Q-learning on the large action set. The agent trained with AQL on the large action set was able to outperform agents trained with the curated action set by navigating more efficiently (e.g. by simultaneously strafing and moving forward). While it may be surprising that AQL achieves better data efficiency than Q-learning on the large action set (where Q-learning evaluates all possible actions), we speculate that AQL benefits from the additional stochasticity in action selection due to approximate maximization, which potentially improves exploration. It is also possible that by using approximate maximization for bootstrapping, AQL is less prone to over-estimation of Q-values, from which Q-learning with exact maximization is known to suffer (van Hasselt, 2010); we leave this investigation to future research. Critically, considering all 3528 actions with exact Q-learning takes about 10 times longer in wall-clock time because the Q-function must be evaluated for each action. IMPALA is also slowed down drastically when considering 3528 actions. These experiments show that AQL can indeed efficiently learn good policies in large discrete structured action spaces and, unsurprisingly, performs comparably to exact Q-learning in low cardinality action spaces.

6 Discussion

We presented AQL, a simple approach to scaling up Q-learning to multi-dimensional action spaces where the sub-actions can be discrete, continuous or a combination of the two. We showed that AQL is competitive with several strong continuous-control methods on low-dimensional control tasks, and is able to outperform them on medium- and high-dimensional action spaces. Perhaps most notably, AQL closes the gap between Q-learning and stochastic actor-critic methods in their ability to handle high-dimensional and structured action spaces. While actor-critic methods have been able to handle such action spaces simply by changing the form of the stochastic policy (Schulman et al., 2015; Mnih et al., 2016), different variants of Q-learning have been developed to handle each specific case, i.e. DDPG for continuous actions and DQN for low-dimensional discrete action spaces. AQL allows one to handle different action spaces simply by varying the form of the proposal distribution while keeping the rest of the algorithm unchanged. While AQL and stochastic actor-critics are equally general in terms of which action spaces they can handle (being limited only by the ability to efficiently sample from a distribution over actions), AQL may have some advantages over actor-critic methods. Namely, AQL supports off-policy learning without the need for importance sampling-based correction terms used by actor-critic methods, making the method easier to implement in replay-based agents.

There are several promising avenues for future work. AQL could be combined with the cross-entropy method (CEM) by modeling the parameters of the initial distribution of CEM with the proposal distribution. Another interesting direction would be to use the proposal distribution for more intelligent exploration than epsilon-greedy exploration. Currently AQL uses a simple way of learning the proposal based on supervised learning. Better ways of optimizing the proposal parameters could improve the overall algorithm. For example, one natural choice would be to train the proposal by maximizing the expected Q-value of a sample from it using REINFORCE (Williams, 1992). Considering sequential proposal distributions is another interesting possibility. By conditioning each sample from the proposal on all previous samples it may be possible to learn a proposal that achieves the same overall performance using many fewer samples.

Acknowledgements

We thank Marlos C. Machado, Jonathan Hunt and Tim Harley for invaluable feedback on drafts of the manuscript. We additionally thank Jonathan Hunt, Benigno Uria, Siddhant Jayakumar and Catalin Ionescu for helpful discussions. Additional credit goes to Wojtek Czarnecki for creating the DeepMind Lab action set with 3528 actions.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems 30*, pp. 5048–5058. 2017.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *International Conference on Learning Representations (ICLR)*, 2018.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458, 2017.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The Helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Shane Gu, Tim Lillicrap, Ilya Sutskever, and Sergei Levine. Continuous deep q-learning with model-based acceleration. In *Proc. of ICML*, 2016.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. 2017.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Roland Hafner. *Dateneffiziente selbstlernende neuronale Regler*. PhD thesis, Universitt Osnabrck, 2009.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

Jonathan J. Hunt, André Barreto, Timothy P. Lillicrap, and Nicolas Heess. Entropic policy composition with generalized policy improvement and divergence correction. In *Proc. of ICML*, 2019.

Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp. 1094–1099, 1993.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lyTjAqYX`.

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

Tim. Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proc. of ICLR*, 2016.

Sungsu Lim, Ajin Joseph, Lei Le, Yangchen Pan, and Martha White. Actor-expert: A framework for using action-value methods in continuous action spaces. *CoRR*, abs/1810.09103, 2018. URL `http://arxiv.org/abs/1810.09103`.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep RL. *arXiv preprint arXiv:1705.05035*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Remi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

Jason Pazis and Michail G Lagoudakis. Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 793–800. ACM, 2009.

Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pp. 226–232. Elsevier, 1994.

Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 12. ACM, 2017.

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. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pp. 6284–6291, 2018.

R. Rubinstein and D. Kroese. *The Cross-Entropy Method.* Springer-Verlag, 2004.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proc. of ICML*, pp. 1889–1897, 2015.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proc. of ICML*, 2014.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. *arXiv preprint arXiv:1711.08946*, 2017.

T Tieleman and G Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

E. Uhlenbeck and S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

Hado van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

Chris Watkins and Peter Dayan. Q-learning. *Machine learning*, 8, 1992.

Anton Orell Wiehe, Nil Stolt Ansó, Madalina M Drugan, and Marco A Wiering. Sampled policy gradient for learning to play the game agar.io. *arXiv preprint arXiv:1809.05763*, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Naoto Yoshida. Q-networks for binary vector actions. *arXiv preprint arXiv:1512.01332*, 2015.

Appendix

A1 Architecture Details

We employ a distributed reinforcement learning architecture inspired by the IMPALA reinforcement learning architecture (Espeholt et al., 2018). There is one centralized GPU learner batching parameter updates on experience collected by a large number of CPU-based parallel actors. In all our experiments we use 100 parallel actors. All experiments employ the architecture detailed in the main paper.

The state embedding network consists of the identity function for the Control Suite, so there is no parameter sharing between the proposal and the value networks. The state embedding network for DeepMind Lab experiments is set to the convolutional network employed in Espeholt et al. (2018): 3 layers of ResNet blocks followed by a fully connected layer with 256 units and a recurrent LSTM core with 256 cells. The weights of the network are optimized with two optimizers: one for the proposal distribution head and one for the shared state embedding network and Q-value head. The parameters of the shared state embedding network are considered constant when optimizing the proposal loss. The actors use a local FIFO replay buffer from which unrolls are sampled without prioritization.

A2 Experimental Details

The actors send the current trajectory to the learner queue at the end of an unroll along with two samples from the local replay buffer. Each local replay buffer can store 10^5 steps, representing a total effective replay size of 10^7 since 100 parallel actors were used. The unroll length is set to 30 for Control Suite experiments and to 100 for DeepMind Lab experiments. We train both optimizers with Adam (Kingma & Ba, 2014) with a learning rate of $2 \cdot 10^{-4}$, default TensorFlow hyperparameters and mini-batches of size 32. We could use a target Q-network as described in the main text, but we do not since we did not observe significant gain in any of our hyperparameter iterations. The discount factor γ is set to 0.99. We use the $Q(\lambda)$ variant due to Peng & Williams (1994) (also Chapter 7, Sutton & Barto (1998)) to compute Q-targets with $\lambda = 0.8$. Following Mnih et al. (2016) and Horgan et al. (2018), we use a different amount of ϵ-greedy exploration for each actor, as this has been shown to improve exploration. The first 10 actors use $\epsilon = 0.5$ and the remaining actors use $\epsilon = 0.05$.

A3 Control Suite

We have uploaded a video of the final performance of the discretized AQL agent for all Control Suite tasks at `https://youtu.be/WgTXjJhe6iQ`. The video shows the behavior of the greedy policy along with the proposal distribution and the Q-values of the sampled proposal actions. The videos are selected by picking the seed with the best performance after training. Closer inspection of the trained proposal distributions of the AQL agent reveals that the sub-action proposal distributions have a general tendency to alternate between near-deterministic, low-entropy distributions at critical decision times and high-entropy distributions. The proposal distributions typically still have high entropy when the task is solved and there is a clear optimal action to maintain the equilibrium. Examples of this behavior can be found in the `finger`, `ball-in-cup`, `hopper`, `humanoid`, and `reacher` tasks. One notable example of the side effect of having high entropy in the proposal distribution can be seen in the `walker-stand` task. The high entropy of the policy results in alternating balancing and walking behavior. Figure 4 visualizes the proposal distribution for the `pendulum-swingup` task. The proposal distribution has low entropy when swinging the pendulum up and low entropy during the balancing stage of the task.

We also considered a simpler AQL method on the Control Suite. The *independent* AQL method models the different sub-actions as conditionally independent given the state as opposed to the variant from the main text where an order over sub-actions is assumed and each sub-action is conditioned on the state as well as preceding sub-actions. Figure 5 shows that the independent, discretized variant performed only slightly worse on average on the Control Suite. The AQL implementation with a continuous proposal distribution used an autoregressive policy. We have also uploaded a video of the final performance of the independent proposal AQL agent for all Control Suite tasks on

Algorithm 2: AQL (DISTRIBUTED)

```
procedure ACTOR
    Input  : Proposal network parameters $\theta^{\mu}$,
             $Q$ network parameters $\theta^{Q}$,
             number of actions to draw from the proposal $N$,
             number of actions to draw uniformly $M$,
             unroll length $T$,
             number of replay steps $R$,
             exploration probability $\epsilon$

    Initialize local replay buffer $\mathcal{R}$.
    repeat
        for $t \leftarrow 1 \ldots T$ do
            Observe state $s_t$
            $A_U := \{\mathbf{a}_j^U\}_{j=1}^M,\ \mathbf{a}_j^U \sim \mathrm{Uniform}(\mathcal{A}_1 \times \ldots \times \mathcal{A}_D)$
            $A_\mu := \{\mathbf{a}_i^\mu\}_{i=1}^N,\ \mathbf{a}_i^\mu \sim \mu(\mathbf{a}|s_t;\theta^\mu)$
            $\mathbf{a}_t^* := \arg\max_{\mathbf{a} \in A_U \cup A_\mu} Q(s_t, \mathbf{a}; \theta^Q)$
            with probability $\epsilon$,
                $\mathbf{a}_t := \mathbf{a}_1^U$ // Select $\mathbf{a}_t$ uniformly at random

            otherwise
                $\mathbf{a}_t := \mathbf{a}_t^*$
            $r_t, s_{t+1} \sim p_E(r_t, s_{t+1}|s_t, \mathbf{a}_t)$ // Take action $\mathbf{a}_t$, receive reward $r_t$

        end
        Send unroll $(s_{1:T}, \mathbf{a}_{1:T}, \mathbf{a}_{1:T}^*, r_{1:T})$ to the learner.
        Add unroll $(s_{1:T}, \mathbf{a}_{1:T}, \mathbf{a}_{1:T}^*, r_{1:T})$ to $\mathcal{R}$.
        for $i \leftarrow 1 \ldots R$ do
            Sample unroll $(s_{1:T}, \mathbf{a}_{1:T}, \mathbf{a}_{1:T}^*, r_{1:T})$ from $\mathcal{R}$.
            Send unroll $(s_{1:T}, \mathbf{a}_{1:T}, \mathbf{a}_{1:T}^*, r_{1:T})$ to the learner.
        end
        Poll the learner periodically for updated values of $\theta^\mu$, $\theta^Q$.
        Reset the environment if the episode has terminated.
    until termination

procedure LEARNER
    Input  : Batch size $B$
    repeat
        Assemble batch of experience $\mathcal{B} = \{(s_{1:T}^b, \mathbf{a}_{1:T}^b, {\mathbf{a}_{1:T}^*}^b, r_{1:T}^b)\}_{b=1}^B$
        Update $\theta^Q$ with a step of gradient descent on
        $\mathcal{L}(\theta^Q) = \sum_{b=1}^B \sum_{t=1}^{T-1} \left[ \left( r_t^b + \gamma Q(s_{t+1}^b, \mathbf{a}_{t+1}^{*,b}; \overline{\theta^Q}) - Q(s_t^b, \mathbf{a}_t^b; \theta^Q) \right)^2 \right]$
        Update $\theta^\mu$ with a step of gradient descent on
        $\mathcal{L}(\theta^\mu) = \sum_{b=1}^B \sum_{t=1}^T \left[ -\log \mu(\mathbf{a}_t^{*,b}(s_t^b)|s_t^b; \theta^\mu) - \lambda H(\mu(\mathbf{a}_t^b|s_t^b; \theta^\mu)) \right]$
        Periodically set $\overline{\theta^Q} = \theta^Q$
    until termination
```

`https://youtu.be/9YIujaHjsQY`. The video shows the behavior along with the proposal distribution and the Q-values of the sampled proposal actions.

Figure 6 shows the mean final performance results for the low-dimensional Control Suite tasks, with 1 or 2 sub-actions. IMPALA and D3PG represent the weakest baselines on these tasks. The mean final performance results for the medium-dimensional Control Suite tasks, with 4 to 6 sub-actions, is shown in Figure 7. For medium-dimensional tasks, D3PG and IMPALA again stand out as the methods that perform worse than the other baselines on average. Results for high-dimensional

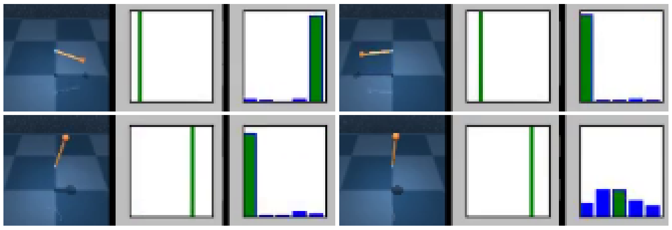

Figure 4: Chronological visualization of the proposal distribution for the `pendulum-swingup` task. The top two images show the pendulum during the swingup phase and the bottom two images visualize the balancing phase. The leftmost part of each plot shows the behavior of the agent. The middle part plots a histogram of Q-values for 100 sampled actions from the proposal distribution with an x-axis that ranges from 0 to 100 (maximum possible value since $\gamma = 0.99$). The right part shows the probabilities of the 5 discretized action options for the proposal distribution. Green bars in the proposal distribution belong to the argmax action and also the selected action since the agent follows the greedy policy. The proposal distribution is near-deterministic while swinging up and near-uniform when it gets close to the balancing equilibrium.

tasks are shown in Figure 8. Here, D3PG and the AQL methods outperform QT-Opt, IMPALA and Uniform Q-Learning.

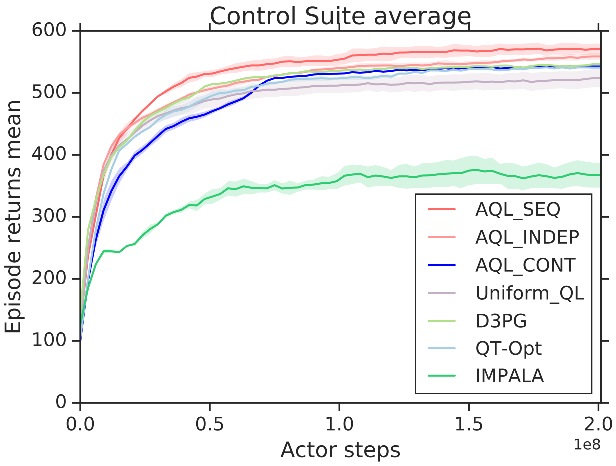

Figure 5: Learning curves of the mean return across all tasks in the Control Suite. The error bars represent the standard error of the mean episode return over 3 seeds. The AQL variant where sub-actions are sampled independently performs slightly worse on average.

Figures 9 and 10 show the individual learning curves of the Control Suite for all tasks.

Somewhat surprisingly, we found that the choice of the network architecture and optimizer along with the optimizer hyperparameters can have a dramatic effect on the final performance. We chose the hyperparameters for the preceding experiments by first running the baselines on some of the high-dimensional tasks for the considered methods and then committing to those settings that were found to work best on average for the all tasks. Figure 11 shows the results of an earlier sweep with an alternative architecture and the RMSProp optimizer (Tieleman & Hinton, 2012) instead of Adam. The architecture in the earlier sweep shared weights in the first two layers and was deeper but had fewer units in each layer. The results of the earlier sweep are significantly worse on average for all baselines except for IMPALA and continuous AQL. The results for continuous AQL are competitive with the best baseline of AQL (discretized, autoregressive proposal distribution), while the IMPALA results using this sharing architecture are comparable to the results from the main text. D3PG was the most affected by the choice of architecture and optimizer type, being the second worst performing

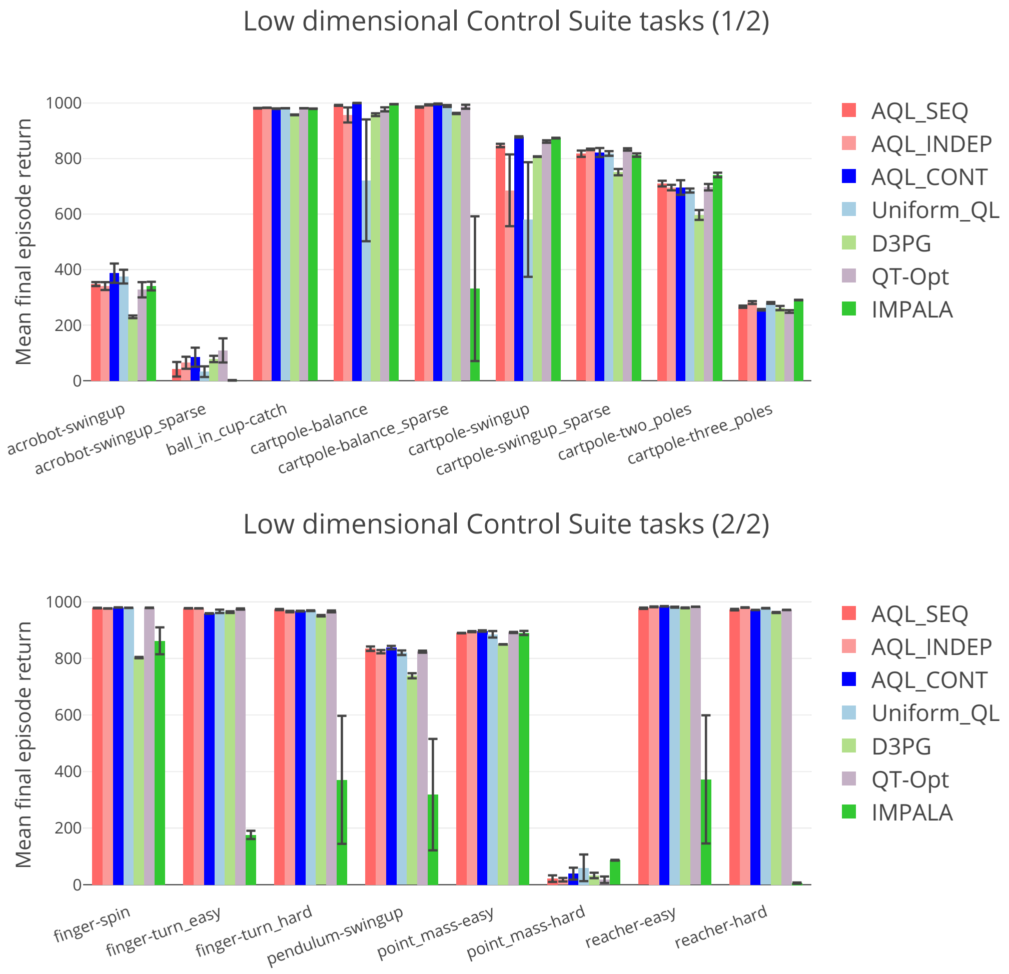

Figure 6: Mean final performance, averaged over 3 seeds, for the low-dimensional Control Suite tasks. The error bars represent the standard error of the mean episode return.

method when trained with RMSProp and the second best performing method when trained with Adam.

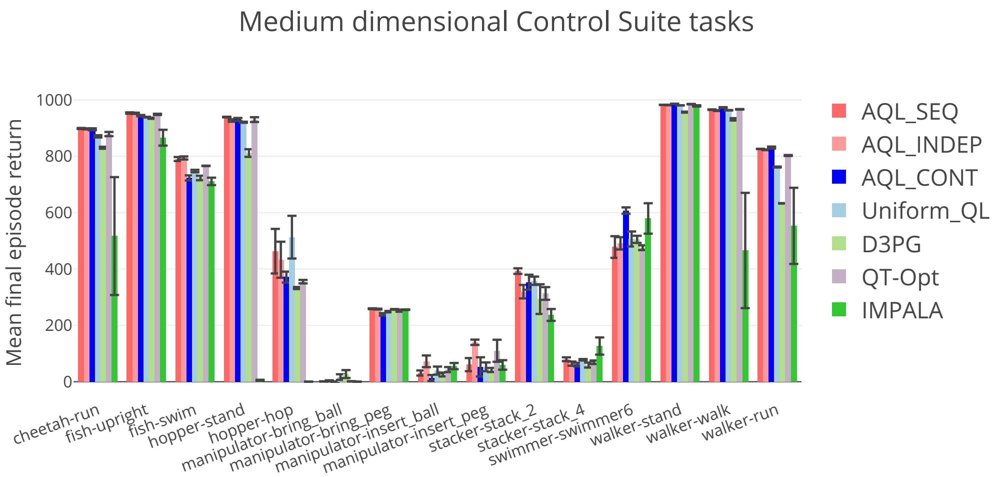

Figure 7: Mean final performance, averaged over 3 seeds, for the medium-dimensional Control Suite tasks. The error bars represent the standard error of the mean episode return.

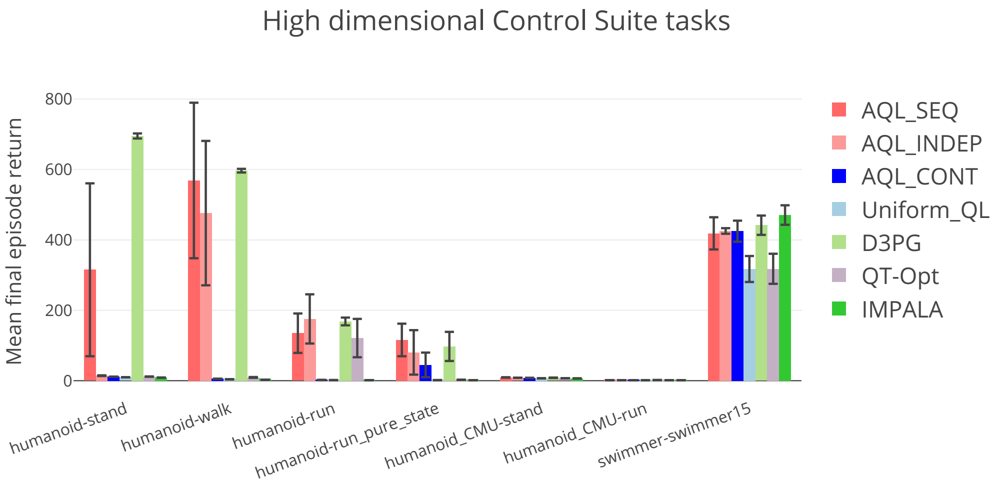

Figure 8: Mean final performance, averaged over 3 seeds, for the high-dimensional Control Suite tasks. The error bars represent the standard error of the mean episode return.

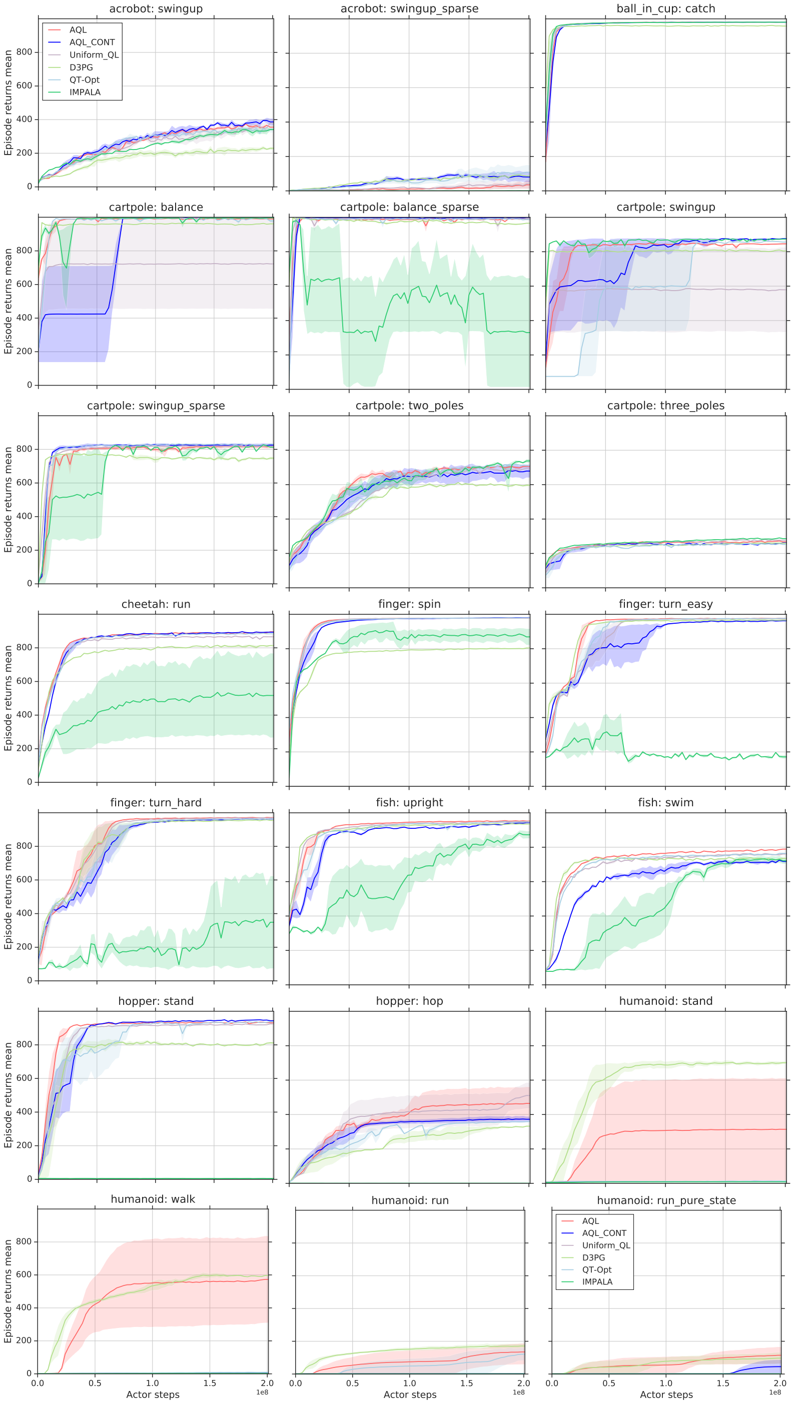

Figure 9: Task specific learning curves for the Control Suite (1/2).

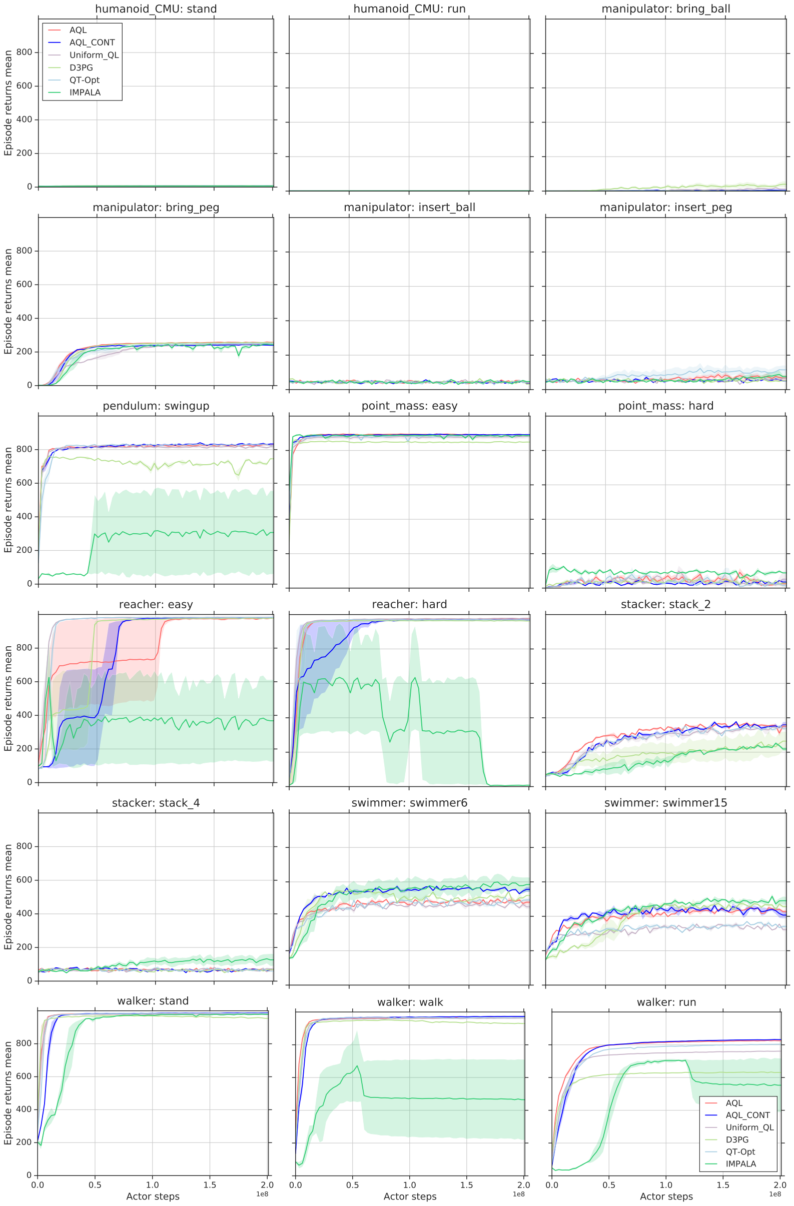

Figure 10: Task specific learning curves for the Control Suite (2/2).

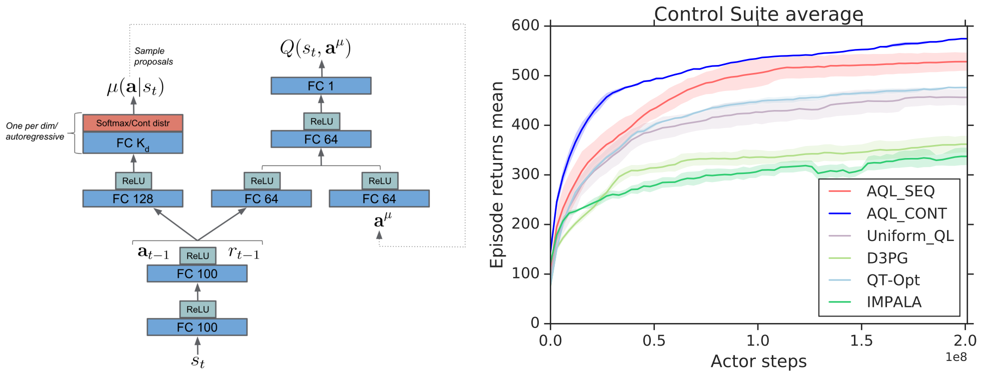

Figure 11: Left: alternative architecture with RMSProp as the optimizer. Right: the corresponding learning curves of the mean return across all tasks in the Control Suite. The error bars represent the standard error of the mean episode return. The results are significantly worse for all baselines compared with the results in the main text.