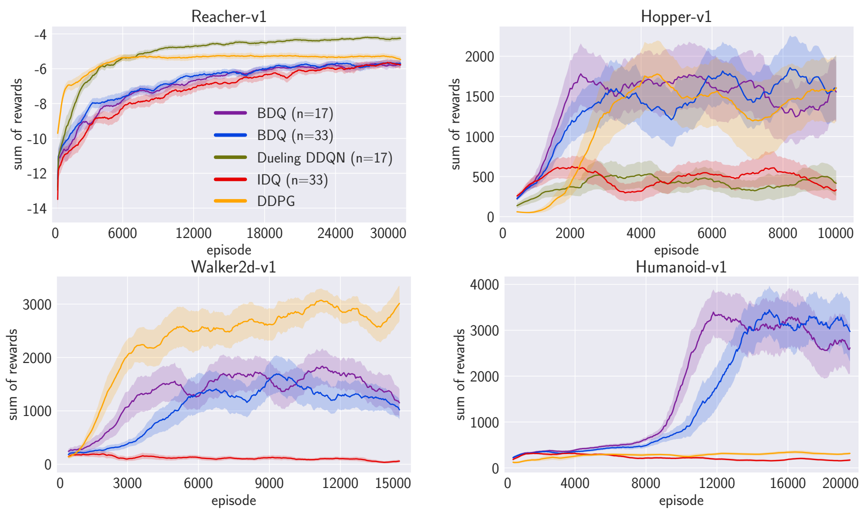

Figure 5: Learning curves for the OpenAI's MuJoCo Gym manipulation and locomotion benchmark domains. The solid lines represent smoothed (window size of 20 episodes) averages over 6 runs with random initialization seeds, while shaded areas show the standard deviations. Evaluations were conducted every 50 episodes of training for 30 episodes with a greedy policy.

work (i.e. $9^5 \approx 6 \times 10^4$) and consequently the extremely large number of network parameters that need to be trained at every iteration. In contrast, in the same task, we see that BDQ performs well and converges to good policies with robustness against the discretization granularity.

Standard Benchmark Environments

Here we evaluate the performance of BDQ on a set of standard continuous control benchmark domains from the OpenAI's MuJoCo Gym collection. Figure **??** demonstrates sample illustrations of the environments used in our experiments. Table **??** states the dimensionality information of these tasks, provided for the specific case of $n = 33$ being the finest granularity we experimented with.

We compare the performance of BDQ against a state-of-the-art continuous-action reinforcement learning algorithm, DDPG, as well as against a completely independent agent, IDQ. For all environments, we evaluate the performance of BDQ with two different discretization resolutions resulting in $n = 17$ and $n = 33$ sub-actions per degree of freedom. We do this to compare the relative performance of BDQ for the same environments with substantially larger discrete action spaces. Where feasible (i.e. Reacher-v1 and Hopper-v1), we also run the Dueling DDQN agent with $n = 17$.

The results demonstrated in Figure **??** show that IDQ's performance quickly deteriorates with increasing action dimensionality, while BDQ continues to perform competitively against DDPG. Interestingly, BDQ significantly outperforms DDPG in the most challenging domain, the Humanoid-v1 task which involves 17 action dimensions, leading to a combinatorial action space of approximately 6.5×10^{25} possible actions for $n = 33$. Our ablation study on BDQ (with a shared network module) and IDQ (no shared network module) verifies the significance of the shared decision module in coordinating the distributed policies, and thus enabling the BDQ agent to progress in learning and to converge to good policies in a stable manner. Furthermore, remarkably, to perform competitively against a state-of-the-art continuous control algorithm in such high-dimensional domains is a feat previously considered intractable for discrete-action algorithms (**?**; **?**).

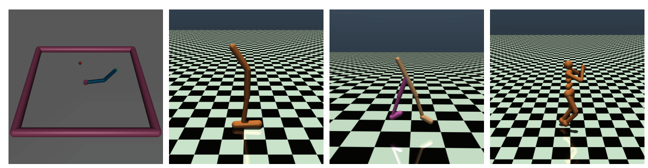

Figure 6: Illustrations of the domains from the OpenAI's MuJoCo Gym that were used in our experiments. From left: Reacher-v1, Hopper-v1, Walker2d-v1, and Humanoid-v1 featuring 2, 3, 6, and 17 degrees of freedom, respectively.

However, in the simpler tasks DDPG performs better or on par with BDQ. We think a potential explanation for this could be the use of a specialized exploration noise process by DDPG which, due to its temporally correlated nature, enables effective exploration in domains with momentum.

By comparing the performance of BDQ for $n = 17$ and $n = 33$, we see that, despite the significant difference in the total number of possible actions, the proposed agent continues to learn rather efficiently and converges to similar final performance levels. An interesting point to note is the exceptional performance of Dueling DDQN for $n = 17$ in Reacher-v1. Yet, increasing the action dimensionality by only one degree of freedom (from $N = 2$ in Reacher-v1 to $N = 3$ in Hopper-v1) renders the Dueling DDQN agent ineffective. Finally, it is noteworthy that BDQ is highly robust against the specifications of the TD target and loss function, while it highly deteriorates with the ablation of the prioritized replay. Characterizing the role of the prioritized experience replay, in stabilizing the learning process for action

Domain	dim($\boldsymbol{o}$)	N	n^N	$n \times N$
Reacher-v1	11	2	1.1×10^3	66
Hopper-v1	11	3	3.6×10^4	99
Walker2d-v1	17	6	1.3×10^9	198
Humanoid-v1	376	17	6.5×10^{25}	561

Table 1: Dimensionality of the OpenAI's MuJoCo Gym benchmark domains: dim($\boldsymbol{o}$) denotes the observation dimensions, N is the number of action dimensions, and n^N indicates the number of possible actions in the combinatorial action space, with n denoting the fixed number of discrete sub-actions per action dimension. The rightmost column indicates the total number of network outputs required for the proposed action branching architecture. The values provided are for the most fine-grained discretization case of $n = 33$.

branching networks, remains the subject of future research.

Experiment Details

Here we provide information about the technical details and hyperparameters used for training the agents in our experiments. Common to all agents, training always started after the first 10^3 steps and, thereafter, we ran one step of training at every time step. We did not perform tuning of the reward scaling parameter for either of the algorithms and, instead, used each domain's raw rewards. We used the OpenAI Baselines (**?**) implementation of DQN as the basis for the development of all the DQN-based agents.

BDQ We used the Adam optimizer (**?**) with a learning rate of 10^{-4}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We trained with a minibatch size of 64 and a discount factor $\gamma = 0.99$. The target network was updated every 10^3 time steps. We used the rectified non-linearity (or ReLU) (**?**) for all hidden layers and linear activation on the output layers. The network had two hidden layers with 512 and 256 units in the shared network module and one hidden layer per branch with 128 units. The weights were initialized using the Xavier initialization (**?**) and the biases were initialized to zero. A gradient clipping of size 10 was applied. We used the prioritized replay with a buffer size of 10^6, $\alpha = 0.6$, and linear annealing of β from $\beta_0 = 0.4$ to 1 over 2×10^6 steps.

While an ϵ-greedy policy is often used with Q-learning, random exploration (with an exploration probability) in physical, continuous-action domains can be inefficient. To explore well in physical environments with momentum, such as those in our experiments, DDPG uses an Ornstein-Uhlenbeck process (**?**) which creates a temporally correlated exploration noise centered around the output of its deterministic policy. The application of such a noise process to discrete-action algorithms is, nevertheless, somewhat nontrivial. For BDQ, we decided to sample actions from a Gaussian distribution with its mean at the greedy actions and with a small fixed standard deviation throughout the training to encourage life-long exploration. We used a fixed standard deviation of 0.2 during training and zero during evaluation. This exploration strategy yielded a mildly better performance as compared to using an ϵ-greedy policy with a fixed or linearly annealed exploration probability. For the custom reaching domains, however, we used an ϵ-greedy policy with a linearly annealed exploration probability, similar to that commonly used for Dueling DDQN.

Dueling DDQN We generally used the same hyperparameters as for BDQ. The gradients from the dueling streams were rescaled by $1/\sqrt{2}$ prior to entering the shared feature module as recommended by **?** (**?**). Same as the reported best performing agent from (**?**), the average aggregation method was used to combine the state value and advantages. We experimented with both a Gaussian and an ϵ-greedy exploration policy with a linearly annealed exploration probability, and observed a moderately better performance for the linearly annealed ϵ-greedy strategy. Therefore, in our experiments we used the latter.

IDQ Once more, we generally used the same hyperparameters as for BDQ. Similarly, the same number of hidden layers and hidden units per layer were used for each independent network, with the difference being that the first two hidden layers were not shared among the several networks (which was the case for BDQ). The dueling architecture was applied to each network independently (i.e. each network had its own state-value estimator). This agent serves as a baseline for investigating the significance of the shared decision module in the proposed action branching architecture.

DDPG We used the DDPG implementation of the rllab suite (**?**) and the hyperparameters reported by **?** (**?**), with the exception of not including a L_2 weight decay for Q as opposed to the originally proposed penalty of 10^{-2} which deteriorated the performance.

Conclusion

We introduced a novel neural network architecture that distributes the representation of the policy or the value function over several network branches, meanwhile, maintaining a shared network module for enabling a form of implicit centralized coordination. We adapted the DQN algorithm, along with several of its most notable extensions, into the proposed action branching architecture. We illustrated the effectiveness of the proposed architecture in enabling the application of a currently restricted discrete-action algorithm to domains with high-dimensional discrete or continuous action spaces. This is a feat which was previously thought intractable. We believe that the highly promising performance of the action branching architecture in scaling DQN and its potential generality evoke further theoretical and empirical investigations.

Unde doloremque quod facere, minus alias harum, ratione possimus est velit eum error a, aperiam itaque quibusdam vel cumque id impedit nam autem nihil, dolorem quas harum omnis libero.Error iste maxime quo nostrum id, cumque numquam sed, perferendis consectetur illum impedit doloremque autem vero voluptas pariatur, ducimus reprehenderit porro dolorum rem possimus laborum sit a quibusdam?Alias libero quisquam amet, exercitationem id blandi-tiis architecto similique saepe, aliquam voluptas