NEURAL DISCRETE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Designing effective action spaces for complex environments is a fundamental and challenging problem in reinforcement learning (RL). Some recent works have revealed that naive RL algorithms utilizing well-designed handcrafted discrete action spaces can achieve promising results even when dealing with high-dimensional continuous or hybrid decision-making problems. However, elaborately designing such action spaces requires comprehensive domain knowledge. In this paper, we systemically analyze the advantages of discretization for different action spaces and then propose a unified framework, Neural Discrete Reinforcement Learning (NDRL), to automatically learn how to effectively discretize almost arbitrary action spaces. Specifically, we propose the Action Discretization Variational AutoEncoder (AD-VAE), an action representation learning method that can learn compact latent action spaces while maintain the essential properties of original environments, such as boundary actions and the relationship between different action dimensions. Moreover, we uncover a key issue that parallel optimization of the AD-VAE and online RL agents is often unstable. To address it, we further design several techniques to adapt RL agents to learned action representations, including latent action remapping and ensemble Q-learning. Quantitative experiments and visualization results demonstrate the efficiency and stability of our proposed framework for complex action spaces in various environments.

1 INTRODUCTION

Recent advances in Reinforcement Learning have yielded many promising research achievements Vinyals et al. (2019); Berner et al. (2019); Schrittwieser et al. (2019). However, the complexity of action spaces still prevents us from directly utilizing advanced RL algorithms to real-world scenarios, such as high-dimensional continuous control in robot manipulation Lillicrap et al. (2016) and structured hybrid action decision-making in strategy games Kanervisto et al. (2022). Complex action spaces lead to extensive challenges in designs of policy optimization Xiong et al. (2018b), efficiency of exploration Seyde et al. (2021b) and behaviour stability of learned agents Bester et al. (2019).

To handle these issues, some existing work first elaborately design particular reinforcement learning methods in original complex action spaces. Specifically, deterministic policy gradient methods Lillicrap et al. (2016); Fujimoto et al. (2018) are designed to handle continuous control problems. And Xiong et al. (2018b); Fan et al. (2019b) propose some techniques to extract the relationship between different action dimensions, which is important in hybrid action spaces. However, these designs often suffer from low exploration efficiency and unstable training due to the infinite action spaces and interference between different sub-actions Bester et al. (2019), respectively. Action space shaping Kanervisto et al. (2020) is another way to tackle these problems. Particularly, many RL applications Kanervisto et al. (2022); Wei et al. (2022) design specific action discretization mechanisms to simplify the decision-making spaces, leading to the promising performance improvement, but it requires intensive investigations about the corresponding environments. Moreover, the combination of many manually discretized sub-actions will result in the exponential explosion of action numbers, which is incompatible with large action spaces. Recently, some works propose to learn abstract action representations to boost RL training. HyAR Li et al. (2021) designs a special training scheme with VAE Kingma & Welling (2014) to map the original hybrid action space to a continuous latent action space. Some other methods Dadashi et al. (2022); Shafiullah et al. (2022); Jiang et al. (2022) build prior sets of discrete actions to from expert demonstrations, and then deploy RL agents on this fixed discrete action sets. To preserve the necessary attributes of environments, all the above discretiza-

tion techniques require related domain knowledge to discard redundant information about actions, which means that they are unsuitable for different environments with arbitrary action spaces.

In this paper, we focus on how to learn a unified discrete action representations from scratch without any domain knowledge. Based on previous analysis and our investigations (as shown in Figure 1), we summarize the following advantages of discretization for the complexity of the action space:

- Unified action discretization provides a powerful and general approach to dealing with reinforcement learning in complex action spaces. It is equivalent to split the entire pipeline into two parts: (1). representation learning and (2). decision-making. The former focus on intrinsic properties and data distributions of the action space, then transform various action spaces into standard discrete action sets, while the latter only needs to solve core decision-making problems.
- Effective discretization can improve sample efficiency by reducing the overhead in repeating sub-optimal, useless, and semantically similar actions. RL agent can just explore and exploit the necessary subsets of the original action space during training.

Then, we introduce Neural Discrete Reinforcement Learning (NDRL) framework. Specifically, inspired by VQ-VAE van den Oord et al. (2017), we propose a action representation method called Action Discretization Variational Auto-Encoder (AD-VAE) to learn latent discrete action space from the original environment, and conduct RL on the learned space utilizing any classical RL techniques about the discrete action. It is essential to capture the intrinsic properties of the original action space, which is beneficial to learn a compact latent action space while keeping necessary information of the original action space. Therefore, we design a state-conditioned action encoder and decoder, and utilize graph neural network (Kipf & Welling, 2016) and soft-argmax operation Luvizon et al. (2019) to improve the capability of AD-VAE for the relationships between different action dimensions and boundary action values. Furthermore, we find a core issue of parallel optimization of AD-VAE and RL agents: the online updates of AD-VAE may lead to semantic changes of latent actions (i.e. the non-stationary of decision spaces), resulting in severe data staleness and Q-value over-estimation. To solve this problem, we introduce action remapping and ensemble Q-learning. Concretely, we apply the classic DQN as an instance to our framework, named **A**ction **D**iscretization **Q**-learning (**ADQ**), which can be deployed for most complex action spaces. Compared with pioneer works (Chandak et al., 2019a; Zhou et al., 2020; Dadashi et al., 2022), to our best knowledge, our proposed framework is the first online RL paradigm capable of employing in discrete action spaces learned from different continuous and hybrid decision-making environments.

To demonstrate the efficiency and stability of our NDRL framework and AD-VAE method, we evaluate it on the classic continuous control benchmark MuJoCo Todorov et al. (2012), showing that ADQ can achieve excellent performance operating in high-dimensional continuous space even with a small number of actions. To evaluate the generality, we test it on the hybrid action environments Gym Hybrid thomashirtz (2021), HardMove from HyAR and GoBigger Zhang (2021). The results show that ADQ outperforms current state-of-the-art hybrid action algorithms in both sample efficiency and final performance. Besides, we also conduct a series of ablation study experiments and interpret more details about NDRL by visualization on the latent space.

2 Related work

Action Discretization Discretization and continuity are like the relationship between 0 and 1 in the binary world. All things, including time and space, are continuous, but for the convenience of cognition, we will discretize all of them. Only then can we have measures, such as the concepts of hours, minutes and meters. In RL, learning directly on a high-dimensional continuous action space may present difficulties in exploration due to the uncountable set of actions. In addition, (Bjorck et al., 2021) argues that the nonlinear function saturation caused by unstable network parameterization will cause the well-known high variance problem. The most straightforward solution is discretization, however, this usually suffers from the curse of dimensionality. To alleviate this problem, many assumptions about the action space have been proposed. For example, (Tang & Agrawal, 2020) verifies the feasibility of discretizing the action space in on-policy optimization by utilizing the factorized distribution across action dimensions. In (Dadashi et al., 2022), the authors proposed to circumvent the curse of dimensionality problem by learning a set of plausible discrete actions from expert demonstrations. We argue that this algorithm can naturally be seen as a special case of

our NDRL framework. (Seyde et al., 2021a) explored the effect of extreme actions on continuous action control, which also inspired the design of AD-VAE for maintaining boundary actions.

Hybrid Action Space Many real-world problems may have hybrid action spaces. For example, in the GoBigger game, we need to select an action type first, and then give its corresponding continuous control arguments. The simplest idea is to map it onto a unified homogeneous action space, like discretizing continuous actions or making discrete actions continuous, however this may create scalability issues with the curse of dimensionality. Going a step further, recent work proposes various hand-designed network structures to learn directly on the original hybrid action space. For instance, Parameterized Action DDPG (Hausknecht & Stone, 2016) uses a modified DDPG actor-critic structure and HPPO (Fan et al., 2019a) proposes different types of heads for different action types. PDQN (Xiong et al., 2018a) and MPDQN (Bester et al., 2019) use a hybrid structure of DQN and DDPG, explicitly modeling the dependencies between continuous and discrete sub-actions.

Action Representation Learning The concept of the latent space is widely used in various elements of reinforcement learning, such as latent state and dynamics. But in action space, (Chandak et al., 2019b) proposes action representation learning in a large action space, leveraging the structure in the space of actions and showing its importance for enhancing generalization over large action sets in real-world large-scale applications. (Li et al., 2021) propose Hybrid Action Representation (HyAR) to learn a compact and decodable latent representation space for the original hybrid action space. HyAR constructs the latent space and embeds the dependence between discrete action and continuous arguments via an embedding table and conditional Variational Auto-Encoder (VAE).

3 BACKGROUND

Markov Decision Process In reinforcement learning, we model a decision-making problem as a Markov Decision Process (MDP) $\mathcal{M}$=$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \rho_0)$, where $\mathcal{S}$ and $\mathcal{A}$ represent the state space and the action space, $\mathcal{P}$ is the transition function: $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$, $\mathcal{R}$ is the expected reward function: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $\gamma \in [0, 1)$ is the discounted factor, and ρ_0 is the initial state distribution. The objective of RL is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ to maximize the expected discounted return $J(\pi) = \mathbb{E}_{\pi,\rho_0,\mathcal{P},\mathcal{R}}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where the expectation is taken with respect to the trajectory distribution induced by π and environment dynamics.

Hybrid Action Space In decision-making problems, at each time t, the agent receives a state and carries out an action, which can be divided into 3 types: discrete, continuous and hybrid action. Here we give a general formalization. A hybrid action a contains N decision nodes (sub-actions). At each decision node i and time step t, the agent needs to give a *proto-action* $a_{t,i}$ with two attributes including type and range of values; the type of value $a_{t,i,type}$ indicates whether it is continuous or discrete, and the range of value $a_{t,i,range}$ indicates its corresponding executable action set. Thus, we use an ordered tuple like $a = (a_{t,1}, a_{t,2}, ..., a_{t,N})$ to describe these basic nodes. Furthermore, the relations between decision nodes can be defined by a adjacency matrix $A_{r,t}$ in graph theory, called action relation matrix. The value of the elements in the matrix is $\{0, 1\}$. If the element $A_{i,j}$ in row i and column j is equal to 1, it means that there is a directed edge from decision node i to j. If equal to 0, there is no dependency between them. In many real-world problems, the dependencies between the *proto-action* always are invariant, that is, the action relation matrix is independent of t. Generally, hybrid action space can be defined as a tuple:

$$\mathcal{A} = (\{a_{t,i,type}, a_{t,i,range} \mid i \in [1, ..., N]\}, A_r) \tag{1}$$

The Parameterized Action Space defined in (Masson et al., 2016) is a special instance of our definition, specifically, which is equivalent to action a containing 2 *decision nodes* and $a_{t,0,type} = 0$, $a_{t,0,range} = \mathcal{K}$, $a_{t,1,type} = 1$, $a_{t,1,range} = \mathcal{X}$. There is only a directed edge from *decision node* 1 to *decision node* 2, formally, the adjacency matrix A_r is:

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{2}$$

Action Transformed MDP Here we augment a MDP with action transformation, which can be defined as $\mathcal{M}$=$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \rho_0, \mathcal{T})$, where $\mathcal{T}$ denotes the transformation operator on action space, such as action discretization . Denote the transformed action as k, we can describe $\mathcal{T}$ as:

$$\mathcal{T} : k = \mathcal{T}(s, a) \;\; a = \mathcal{T}_{-1}(s, k) \tag{3}$$

The other elements are consistent with the original definition of MDP. Through this transformation, we can learn an RL agent more efficiently on this new, often reduced, latent action space.

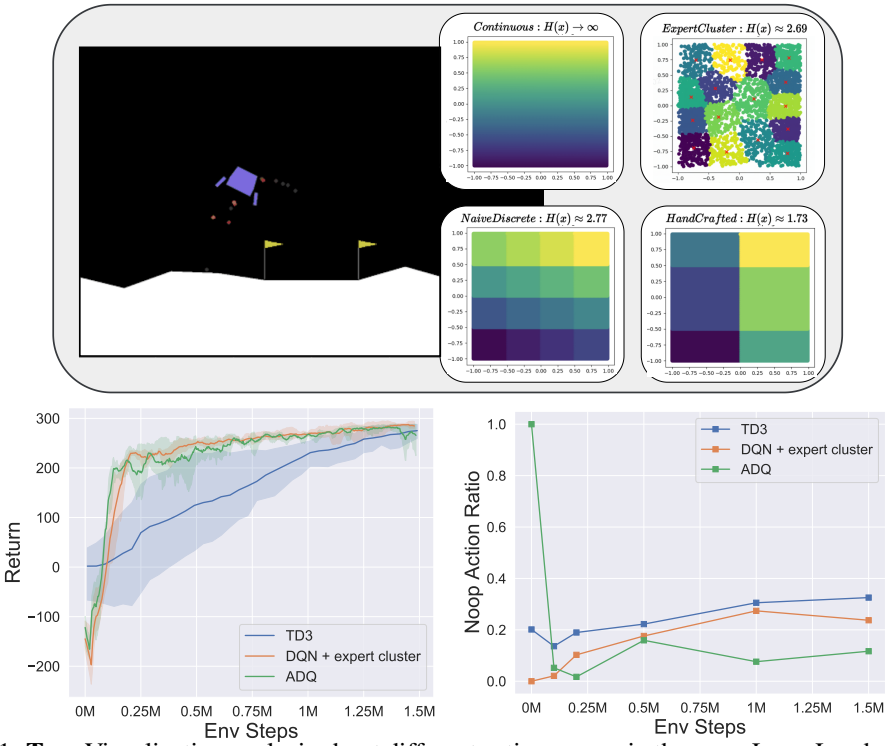


Figure 1: **Top**: Visualization analysis about different action spaces in the same LunarLander (Brockman et al., 2016) environment. *ExpertCluster* is discrete action obtained by clustering on TD3 expert data). *HandCrafted* is obtained by the threshold of spaceship engine. Vertical engine only enables when x is bigger than 0; And if y is smaller than -0.5, the left booster will fire, and if y is bigger than 0.5, the right booster will fire. **Bottom**: (left) Episode return of three algorithms on LunarLander: TD3 (original continuous action space), DQN + expert cluster, ADQ (discrete action learned by AD-VAE from scratch) ; (right) The ratio of some semantically same actions (no operation), more *no-op* actions means greater redundancy during training, i.e., lower is better.

Vector Quantised Variational AutoEncoders Motivated by vector quantization (VQ) and Variational AutoEncoder, VQ-VAE van den Oord et al. (2017)) is designed to learn a discrete latent representations to represent the original data distribution (*e.g.*, text, image) in a unsupervised manner. VQ-VAE mainly comprises of an encoder e_ϕ, an decoder d_ψ, and a learnable code table V_ε. The learnable code table maintains a set of embeddings $\{e_k\}_{k=0}^{K-1}$.

Firstly, the encoder takes the data x as input, and outputs an embedding vector $z^e = f(x)$. Then using the embedding vector z^e to query the nearest (usually in Euclidean distance) code vector z^d in the code table and outputs an latent index k simultaneously. Thirdly, the decoder uses the code vector e^d as its input to produce reconstructions $\hat{x}$. The whole objective is to minimize the following loss function

$$\mathcal{L} = \mathcal{L}_d(\hat{x}, x) + \|\text{sg}\left[e_\phi(x)\right] - z^e\|_2^2 + \beta \left\|e_\phi(x) - \text{sg}[z^e]\right\|_2^2 \tag{4}$$

$$z^d = e_k, \text{ where } k = \text{argmin}_j \left\|z^e - e_j\right\|_2 \tag{5}$$

Where $sg(\cdot)$ is the stop gradient function. The first term of loss function is to reconstruct error in certain distance metric, the second item and the third term is embedding loss and commitment loss respectively. Please refer to van den Oord et al. (2017) for more details.

4 Neural Discrete Reinforcement Learning

4.1 Motivation

First, to motivate our proposed framework, we further discuss the advantages of reinforcement learning in discrete action spaces from the following 3 aspects: unity, efficiency and stability.

4.1.1 UNITY

In practice, researchers need to first transform the original decision-making problem into a standard MDP form. Due to the different types of target action spaces, it is inevitable to utilize different techniques for the corresponding action spaces, which brings non-negligible learning and tuning costs beyond core RL optimization. But when we dive deeper into this problem, we find there are obvious redundancies in most complex action space, e.g., only a few discrete actions/samples can perform well in multi-dimensional continuous control Seyde et al. (2021b); Hubert et al. (2021). In Figure 1(a), we also illustrate the entropy of different action spaces to show the effectiveness of discretization. Therefore, learning compact action representations instead of repeating some dirty work in the raw action space is a natural and powerful choice. Moreover, the decoupling of action representation learning and RL allows researchers to concentrate on only one of the topics.

4.1.2 EFFICIENCY

Furthermore, we verify the efficiency of action discretization in online RL training. As shown in Figure 1(a), we find the well-designed discrete action space shows lower entropy and more compact representation. We also conduct a simple experiment to test the performance and the ratio of useless action like excessive *no operation* in 1(b). Based on these observations, we design AD-VAE to automatically learn the latent discrete action space from the original action space. On one hand, this model can learn to approximate the necessary parts and ignore meaningless parts of the original space, which is beneficial to improve exploration efficiency. On the other hand, some works Jiang et al. (2022) show that the marginal distribution of each action dimension is often multi-modal, thus using a discrete categorical distribution is more suitable than the simple regression in TD3.

4.1.3 STABILITY

However, previous methods Dadashi et al. (2022); Jiang et al. (2022) only succeed in deploying action discretization on imitation learning and offline RL settings, which means that it needs to first learn a latent space and then apply decision-making algorithms on the fixed discrete action space. We tried to directly utilize AD-VAE on online settings but obtain unstable episode returns. Compared to training on the frozen discrete action space, we find some abnormal indicators including unusually high Q-values and obvious fluctuations in gradient scale and variance. Due to parallel optimization about AD-VAE and RL agents, it would be a hazardous non-stationary MDP if the latent action space changes too much. To figure out this problem, we propose action remapping and ensemble Q-learning. Together with AD-VAE, these techniques form the entire NDRL framework.

4.2 NDRL FRAMEWORK

4.2.1 OVERVIEW

Motivated by the above analysis, we propose a framework that combines online RL training with learnable action discretization on complex action spaces, named **N**eural **D**iscrete **R**einforcement **L**earning (NDRL). At a high-level, this framework is a "meta-algorithm" that splits decision-making in arbitrary complex action spaces into two parts: the part of representation learning mapping the original action space to a new discrete action space and the reinforcement learning part built on learned discrete representations. Based on this design, any reinforcement learning designed for discrete space (e.g. DQN, PPO) can be potentially applied to different complex action spaces. The overview of NDRL framework is described in Figure 1, and we will introduce the data collecting phase and network training phase respectively:

Collecting Phase: This phase describes how to collect data used in the training of AD-VAE and RL agents. Given the current state s_t, RL agents first select corresponding discrete latent action k_t, then utilizing AD-VAE decoder to transform it back to the original action space. Moreover, NDRL also deploys some extra randomization operations (e.g. epsilon greedy in DQN) on the decoded action to maintain sufficient exploration. The final action a_t interacts with the environment and it returns reward r_t and next state s_{t+1}. All necessary data will be packed into a transition and put into buffers.

Training Phase: The training phase is to execute two training pipelines with different data buffers in parallel. (1) For action representation learning, NDRL follows the main training scheme of VQ-VAE to reconstruct primitive actions with state conditions. (2) For RL training, it first remaps the

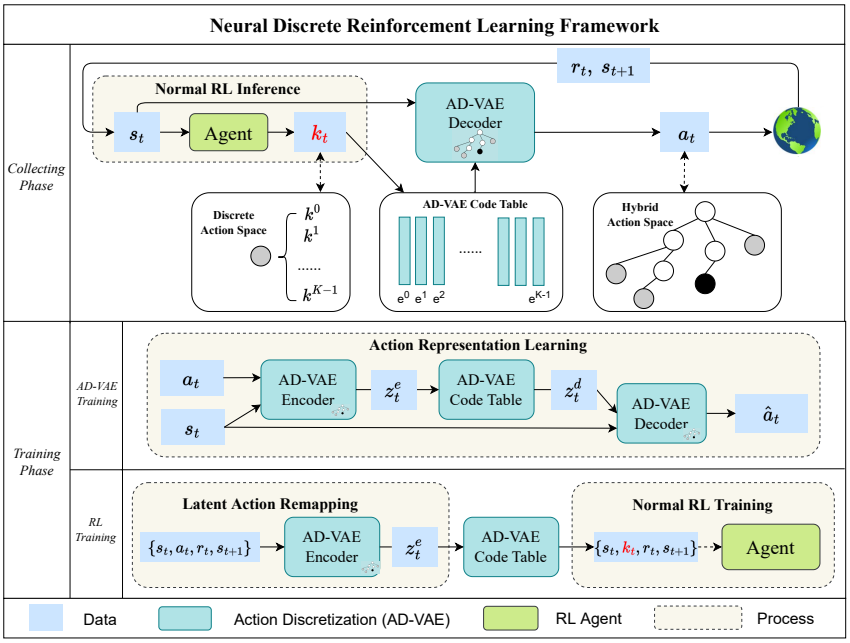


Figure 2: Overview of NDRL framework. **Collect Phase**: Given the current state s_t, the RL agent gives latent discrete action k_t, and utilizes AD-VAE code table and decoder to obtain the raw action a_t, then interacts with the environment. **Train Phase**: First remap the sampled transition (s_t, a_t, r_t, s_{t+1}) with latest AD-VAE encoder and table to k_t, then deploy the normal RL training. Note the AD-VAE is also trained with state condition in parallel. Black round means continuous action space, while grey round means discrete, the whole tree structure means hybrid action.

original action in the sampled transition with AD-VAE encoder to obtain the latest latent action k_t, then conduct normal policy optimization on transformed data.

Besides, for the better initialization of off-policy RL algorithms, we also design a pretrain stage at the beginning of the entire algorithm. The full pseudo-code of NDRL is provided in Algorithm 1. If there are some expert demonstrations, a promising set of discrete action candidates can be learned from it. Otherwise, we can collect data with random policy to train the AD-VAE and learn some basic properties of the original action space for subsequent parallel learning. After pretrain stage, data collecting and two parts of training can be executed asynchronously, so the computational cost of NDRL can be easily optimized and show the same efficiency as other methods.

4.2.2 AD-VAE

In this section, we first analyze the problem of directly using the original VQ-VAE, then introduce the specific design about our proposed Action Discretization Variational AutoEncoders (AD-VAE).

Modeling Intrinsic Properties of Action Spaces In online RL training, there are both necessary and redundant subsets of the original action space. For instance, Seyde et al. (2021b) pointed out that in some continuous action tasks, the optimal action may be at some extreme boundary values (e.g. -1 and 1), which is a common phenomenon in several physical simulation environments. Also, Bester et al. (2019) revealed that effective combinations between different action parts is significant for the optimization in hybrid action space. Therefore, it is wise to pay more attention to those actions that are more beneficial to the optimal policy, and ignore some useless even harmful actions. Otherwise, trivial action reconstruction with the same weights and distance metrics can only obtain some over-smooth actions. Besides, we also can take advantages of the value function to focus on those actions with higher future return, saving the cost of agent exploration and exploitation.

Information Completion in AD-VAE We first introduce the technique of information completion in AD-VAE. In some complex environments, the set of optimal actions could be large and vary in different training stages. If we reconstruct actions without state information, AD-VAE must maintain a large discrete action sets and RL agents needs to learn decisions on a great number of actions, which can easily make it overwhelming and cause training instability. On the contrary, properly use of state-conditioned input in both encoder and decoder of AD-VAE can increase the representational

power and diversity, and reduce the burden of action reconstruction and RL training. Moreover, the relationship between different action parts, i.e. the action relation matrix mentioned in Section 3, can assist AD-VAE to learn more efficiently and reasonably, so we can represent this information as the connections of nodes with a graph neural network. Besides, AD-VAE is not designed to learn the entire action space but to properly model the subsets required by current RL optimization, thus we customize sampling method and training scheme for AD-VAE, including sampling a mixture of stale data in replay buffer and latest collected data as a training mini-batch, validating the reconstruction error of actions to indicate the update frequency.

Continuous Action Regression in AD-VAE Another important intrinsic property of original action spaces is some special action values, such as the extreme actions mentioned in Seyde et al. (2021b) or thresholds of engine dynamics in LunarLander. It is critical for action representation networks to restore these continuous value accurately. Therefore, in AD-VAE, we adopt the soft-argmax operation to automatically reconstruct these special actions. Assuming the range of original action is $[A_{\text{min}}, A_{\text{max}}]$, and it is divided into $N+1$ bins on average, the predicted action is:

$$\hat{a} = \sum_{j=0}^{N} s_j * p_j\left(s_i\right), s_j = A_{\text{min}} + j * (A_{\text{max}} - A_{\text{min}})/N \tag{6}$$

When evaluation, we directly output the corresponding support value if the probability of the support is greater than a threshold (e.g. 0.9). In some environments like *Hopper/Halfcheetah*, we find this design can help a simple DQN agent achieve a comparable performance with TD3.

Other parts of AD-VAE follow the design of VQ-VAE, the whole training procedure is to minimize the loss function described in Equation 4.

4.2.3 Adapting RL to Latent Action Spaces

In this section, we continue to analyze why and how to adapt online RL to latent discrete action spaces, and then illustrate an instance of our NDRL framework on the value-based RL algorithm DQN, Action Discretization Q-learning (ADQ).

Semantic Inconsistency In online RL, Double DQN (van Hasselt et al., 2016) pointed out that Q-value over-estimation problems caused by the function approximation error and the max operator in the bootstrap target often lead to performance deterioration. Furthermore, this problem may be exacerbated in NDRL. On one hand, latent action stored in replay buffer will be stale and biased due to updates of AD-VAE. On the other hand, since AD-VAE is dynamically and simultaneously updated together with RL agents, for a particular latent action, the corresponding action in the original action space could often change, which is more likely to lead to the overestimation of Q-value.

Latent Action Remapping Similar to the reanalyze operation in MuZero Schrittwieser et al. (2019), we design a latent action remapping operation to solve the problem of stale data. In the collected mini-batch $\{s_t, a_t, k_t^{old}, r_t, s_{t+1}\}$, the latent action is determined by the old version of AD-VAE. When updating RL agents, we remap the original action to the corresponding latent action via the latest action encoder e_ϕ: $k_t = e_\psi(s_t, a_t)$, and then executes RL training on the remapped samples $\{s_t, sg[k_t^{new}], r_t, s_{t+1}\}$ (sg means the stop gradient operation).

Ensemble Q-learning To further alleviate the *Semantic Inconsistency* problem, inspired by previous work Anschel et al. (2017); An et al. (2021), we propose an ensemble Q-learning method for more stable Q-value updates for latent action space, reducing the uncertainty of approximation error and over-estimation, which greatly improves the stability of parallel optimization of AD-VAE and RL agents. Specifically, we utilize a shared state encoder and N ensemble Q-value heads, i.e., the penultimate layer of the Q network is connected to N linear layers and outputs N Q-value, then we adjust the update equation of Double DQN as follows:

$$L_i = \left[Q(s_t, a_t; \theta_k) - [r_t + \gamma min_k Q(s_{t+1}, a_{t+1}; \hat{\theta}_k)]\right]^2 \tag{7}$$

$$a_{t+1} = argmax \frac{1}{N} \sum_k Q(s_{t+1}, a_{t+1}; \theta_k) \tag{8}$$

Where L_i means the loss function of *i-th* Q head. Detailed experiments can be found in Section 5.3.

Figure 3: Training curves of ADQ against other baseline algorithms in environments with complex action spaces. **Top**: In four continuous action environments of MuJoCo, ADQ significantly outperforms DQN with manually discretized action space, and is basically comparable to the classic TD3 algorithm. **Bottom**: In four hybrid action environments in Gym-Hybrid, HardMove and GoBigger, ADQ outperforms the baseline MPDQN and HPPO in both performance and stability. Curves and shadings denote the mean and standard deviation over 5 seeds.

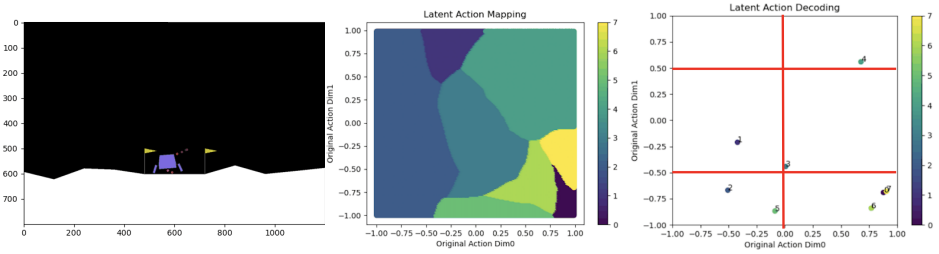


Figure 4: Visualization of the latent action space of LunarLander games. Details is in 5.2.

5 EXPERIMENTS

For the evaluation of our NDRL framework, we ask and answer the following questions: 1) Is it efficient and stable to employ online RL training on discretization action spaces for different decision problems, especially in high-dimensional continuous and hybrid action spaces? (Section 5.1); 2) How do we interpret the training of the latent action space? Can we further verify some observations mentioned in introduction parts? (Section 5.2); 3) How do various designs improve the NDRL framework, including AD-VAE and other RL adaption techniques? (Section 5.3).

5.1 MAIN RESULTS

In this section, we investigate the performance and efficiency of NDRL in various continuous and hybrid action environments against previous algorithms designed specifically for these action spaces. Firstly, we evaluate our methods on MuJoCo, a classic continuous control benchmark, including two high-dimensional continuous domains (Ant and Humanoid with 8 and 17 dimensions respectively). Note we also add a few redundant dimensions in original action spaces. We set up two comparison groups for our ADQ, one is the popular continuous action space algorithm TD3, and the other is naive DQN deployed in the manually discretized action space, i.e., equally dividing the original continuous action into 3 bins at each dimension and using their Cartesian product to obtain handcrafted discrete actions. In Figure 3, ADQ can acquire comparable results to TD3 and show a obvious improvement over naive DQN in all four domains on the top. Secondly, we leverage Gym-Hybrid, HardMove and GoBigger to verify the effectiveness of ADQ in more complex hybrid action spaces. This requires to deal with the relationship between different actions parts (e.g., the value of action arguments depend on the choice of action type) and more complex environment dynamics (GoBigger). At the bottom of Figure 3, we compare ADQ with two types of hybrid action space algorithms, MPDQN and HPPO, suggesting that ADQ can automatically learn the intrinsic properties of discrete action type and continuous action arguments, and performs excellent performance and solid stability. All the detailed settings are shown in Appendix A.1 and A.2 respectively.

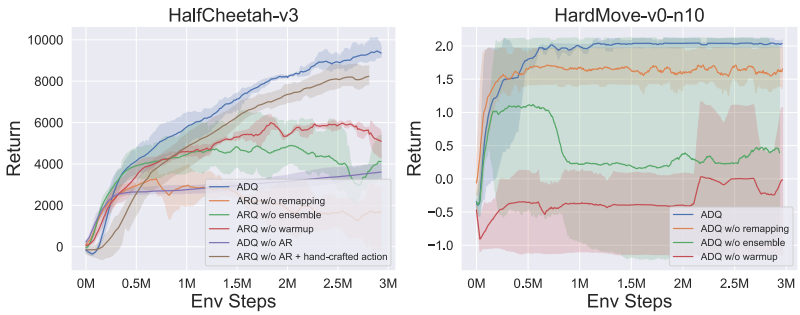


Figure 5: Ablating results of ADQ in two types of environments (continuous and hybrid) over 5 seeds. When we remove any one of the proposed techniques, the performance of ADQ will drop significantly, which verifies the effectiveness of our proposed techniques. *ADQ w/o AR* is especially important for improving ADQ performance in environments such as *HalfCheetah-v3*.

5.2 Visualization Analysis of Latent Action Space

Furthermore, we demonstrate the learned latent action representation in LunarLander environment for 2-dimensional continuous control. Figure 4 first shows the status of the spaceship when it is about to land (left), i.e., launching both horizontal and vertical engines to control speed and position, then uniformly samples points in original continuous space and transforms them with AD-VAE encoder to find their nearest discrete indexes (middle). Also, we directly send the corresponding embeddings in code table to decoder to acquire their counterparts in the raw space (right). We can observe that the learned latent space is similar to the intrinsic mechanisms of this environment, highlighted by red line: only using one discrete action to represent less important actions in this state like *no-op*, mapping several different actions to the bottom right corner.

5.3 Ablation Studies

We also empirically evaluate the specific impacts of our proposed AD-VAE and RL adaption techniques on two example environments respectively: *HalfCheetah-v3* (continuous), and *HardMove-v0-n10* (hybrid). The ablation results are shown in Figure 5. Concretely, we have the following five ablation variants in total, and their brief descriptions are as follows:

ADQ w/o remapping: ADQ variant that does not remap latent actions during RL training.

ADQ w/o ensemble: A variant of ADQ that doesn't utilizing the *Ensemble Q-Learning* technique.

ADQ w/o warmup: ADQ variant that starts training without any warmup pretraining.

ADQ w/o AR: ADQ variant with traditional VQ-VAE reconstruction head.

ADQ w/o AR + hand-crafted action: Built on **ADQ w/o AR**, this variant adds manually selected boundary actions (i.e. the Bernoulli extreme actions) to latent discrete action spaces for RL training.

Figure 5 show that when we remove either of the proposed techniques, the performance of ADQ drops significantly in both two environments, verifying the effectiveness of our proposed techniques. Due to the semantic inconsistency problem, the **ADQ w/o ensemble** agent suffers from severe over-estimation issues and finally show poor performance. The **ADQ w/o remapping** agent meets the same problem, but in some cases the over-estimation problem can be partially alleviated by *Ensemble Q-learning* technique. The **ADQ w/o warmup** agent shows much slow learning progress due to lack of good starting points. Note that in environments with boundary optimal actions such as *HalfCheetah-v3*, **ADQ w/o AR** is especially important for improving ADQ performance, even better than the variant using extra hand-crafted actions. Other ablation results like pretraining on expert demonstrations and the sensity of hyper-parameters can also be found in Appendix.

6 Conclusions and Limitations

Starting from comprehensive analysis for action discretization, we introduce a general and efficient paradigm named Neural Discrete Reinforcement Learning, including our proposed AD-VAE and RL adaption techniques. We empirically evaluate the efficiency and stability of our framework. Although our method achieve superior performance in different benchmark environments, there are still some challenging action spaces in multi-agent games, such as variable-length actions in episodes. Besides, combining latent discrete actions with MCTS is also a valuable attempt. We will continue to pursue ultimate solution for action space shaping in future work.

REFERENCES

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 7436–7447, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/3d3d286a8d153a4a58156d0e02d8570c-Abstract.html.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 176–185. PMLR, 2017. URL http://proceedings.mlr.press/v70/anschel17a.html.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

Craig J. Bester, Steven D. James, and George Dimitri Konidaris. Multi-pass q-networks for deep reinforcement learning with parameterised action spaces. *CoRR*, abs/1905.04388, 2019. URL http://arxiv.org/abs/1905.04388.

Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222*, 2021.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Y. Chandak, G. Theocharous, J. Kostas, S. M. Jordan, and P. S. Thomas. Learning action representations for reinforcement learning. In *ICML*, volume 97, pp. 941–950, 2019a.

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International conference on machine learning*, pp. 941–950. PMLR, 2019b.

Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4537–4557. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/dadashi22a.html.

Z. Fan, R. Su, W. Zhang, and Y. Yu. Hybrid actor-critic reinforcement learning in parameterized action space. In *IJCAI*, pp. 2279–2285, 2019a.

Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. Hybrid actor-critic reinforcement learning in parameterized action space. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 2279–2285. ijcai.org, 2019b. doi: 10.24963/ijcai.2019/316. URL https://doi.org/10.24963/ijcai.2019/316.

S. Fujimoto, H. v. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80, pp. 1582–1591, 2018.

M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4476–4486. PMLR, 2021. URL http://proceedings.mlr.press/v139/hubert21a.html.

Zhengyao Jiang, Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, and Yuandong Tian. Efficient planning in a compact latent action space. *CoRR*, abs/2208.10291, 2022. doi: 10.48550/arXiv.2208.10291. URL https://doi.org/10.48550/arXiv.2208.10291.

Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. Action space shaping in deep reinforcement learning. In *IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020*, pp. 479–486. IEEE, 2020. doi: 10.1109/CoG47356.2020.9231687. URL https://doi.org/10.1109/CoG47356.2020.9231687.

Anssi Kanervisto, Stephanie Milani, Karolis Ramanauskas, Nicholay Topin, Zichuan Lin, Junyou Li, Jianing Shi, Deheng Ye, Qiang Fu, Wei Yang, Weijun Hong, Zhongyue Huang, Haicheng Chen, Guangjun Zeng, Yue Lin, Vincent Micheli, Eloi Alonso, François Fleuret, Alexander Nikulin, Yury Belousov, Oleg Svidchenko, and Aleksei Shpilman. Minerl diamond 2021 competition: Overview, results, and lessons learned. *CoRR*, abs/2202.10583, 2022. URL https://arxiv.org/abs/2202.10583.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Boyan Li, Hongyao Tang, Yan Zheng, Jianye Hao, Pengyi Li, Zhen Wang, Zhaopeng Meng, and Li Wang. Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation. *arXiv preprint arXiv:2109.05490*, 2021.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Diogo C. Luvizon, Hedi Tabia, and David Picard. Human pose regression by combining indirect part detection and contextual information. *Comput. Graph.*, 85:15–22, 2019. doi: 10.1016/j.cag.2019.09.002. URL https://doi.org/10.1016/j.cag.2019.09.002.

W. Masson, P. Ranchod, and G. D. Konidaris. Reinforcement learning with parameterized actions. In *AAAI*, pp. 1934–1940, 2016.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL http://arxiv.org/abs/1911.08265.

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. *Advances in Neural Information Processing Systems*, 34:27209–27221, 2021a.

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin A. Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances*

in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pp. 27209–27221, 2021b. URL https://proceedings.neurips.cc/paper/2021/hash/e46be61f0050f9cc3a98d5d2192cb0eb-Abstract.html.

Nur Muhammad (Mahi) Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *CoRR*, abs/2206.11251, 2022. doi: 10.48550/arXiv.2206.11251. URL https://doi.org/10.48550/arXiv.2206.11251.

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pp. 5981–5988, 2020.

thomashirtz. Gym hybrid. https://github.com/thomashirtz/gym-hybrid, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 6306–6315, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 2094–2100. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z. URL https://doi.org/10.1038/s41586-019-1724-z.

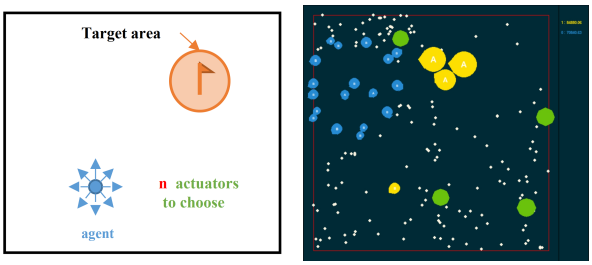
Hua Wei, Jingxiao Chen, Xiyang Ji, Hongyang Qin, Minwen Deng, Siqin Li, Liang Wang, Weinan Zhang, Yong Yu, Lin Liu, Lanxiao Huang, Deheng Ye, Qiang Fu, and Wei Yang. Honor of kings arena: an environment for generalization in competitive reinforcement learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2022.

J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *CoRR*, abs/1810.06394, 2018a.

Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *CoRR*, abs/1810.06394, 2018b. URL http://arxiv.org/abs/1810.06394.

Ming Zhang. Gobigger: A scalable platform for cooperative-competitive multi-agent reinforcement learning. https://github.com/opendilab/GoBigger, 2021.

W. Zhou, S. Bajracharya, and D. Held. PLAS: latent action space for offline reinforcement learning. *CoRR*, abs/2011.07213, 2020.



(a) HardMove-v0 (b) GoBigger

Figure 6: Benchmarks with complex actions: (a) The objective of the agent is to reach the target area. The agent has n equally spaced actuators. It can choose whether each actuator should be on or off and determine the corresponding continuous parameter for each actuator simultaneously. (b) The objective of the agent is to increases its size by colliding and merging with other balls within a bounded rectangular area in a limited time.

A Appendix

A.1 Benchmark Environments

In this section, we provide the brief descriptions for the benchmark environments used in our experiments.

MuJoCo stands for Multi-Joint dynamics with Contact. It is a general-purpose physics engine designed to aid research and development in robotics, bio-mechanics, graphics and animation, machine learning, and other fields that require the rapid and precise modeling of articulated structures interacting with their surroundings Todorov et al. (2012). Features of MuJoCo environment include continuous action spaces and reward representations comprised of many components which usually include penalization of actions corresponding to bad control. We test our proposed ADQ and other baseline algorithms in four MuJoCo environments specifically: *Hopper-v3*, *HalfCheetah-v3*, *Ant-v3*, *Humanoid-v3*.

Gym-Hybrid is a set of sandbox environments for parameterized action-space algorithms. The goal of the agent is to stop inside the target area. The field is a square with a side length of 2. The target area is a circle with radius 0.1. There are three discrete actions: *turn, accelerate, and brake*. In addition to the action, there are 2 possible complementary parameters: *acceleration and rotation*. We also utilize the *HardMove-v0-n* proposed in (Li et al., 2021): The agent has n equally spaced actuators. Agent must choose whether each actuator be on or off (thus 2^n in total) and determine the corresponding continuous parameter for each actuator (moving distance) to reach the target area. The larger n means the larger action space, and the harder it is for the agent to explore and learn. In all our experiments, we set *n=10*.

GoBigger TODO(pu): zhenjie is a multi-agent reinforcement learning environment that emphasizes cooperation and competition. Each agent, which is represented by one or more balls (dubbed clone ball), increases its size by colliding and merging with other balls within a bounded rectangular area in a limited time. The larger the size of clone balls, the higher the player's score. In GoBigger, the observation space includes information about all units in the agent's local field of view. The reward is to take the difference of the sizes in two consequent timesteps. The action space is a hybrid action space *(x, y, action_type)* same as *Gym-Hybrid*. As multi-agents rapidly develop or eliminate opponents through continuous cooperation, cooperation usually requires fine actions to accomplish. So action representation is a great challenge for GoBigger. GoBigger has multiple sub-environments that researchers can design for different tasks. Commonly used environments are *t2p2*, *t3p2*, *t4p3*. Among them, the number behind t (team) means that there are t teams in a game, and the number behind p (player) means that each team contains p agents. In our experiments, we set *t=p=2*.

A.2 Implementation Details

In this section, we provide extensive implementation details for our experimental setup, including the architecture of AD-VAE and Ensemble Q network, hyperparameters, as well as the computational cost in our experiments. Our algorithm pseudocode is shown in .

Algorithm 1: Neural Discrete Reinforcement Learning

Initialize (state-conditioned) AD-VAE: Encoder e_ϕ, Decoder d_ψ, Action embedding table V_ε
Initialize action representation (AR) buffer $\mathcal{D}_{\text{AR}}$, reinforcement learning (RL) buffer $\mathcal{D}_{\text{RL}}$
Initialize RL agent networks (such as Q_θ and/or π_ω)
repeat Stage ❶
  repeat
    Collect data using random/expert policy and store $\{s_t, a_t, r_t, s_{t+1}\}$ in buffer $\mathcal{D}_{\text{AR}}$
  until *reaching maximum warm-up collecting steps*;
  repeat
    Sample mini-batch $\{s_t, a_t\}$ from $\mathcal{D}_{\text{AR}}$
    Update ϕ, ψ and ε using the sampled mini-batch
  until *reaching maximum warm-up AT training steps*;
until *reaching maximum warm-up environment steps*;
clear AR buffer $\mathcal{D}_{\text{AR}}$
repeat Stage ❷
  // agent interacting phase
  for $t \leftarrow 1$ *to* T **do**
    // select latent action in latent action space
    $k_t = \chi_{\text{RL}}(\text{agent}(s_t))$ ▷ agent represents the value or policy network associated with the RL algorithm.
    // decode into original action space
    $a_t = \chi_{\text{AR}}(\bar{a}_t)$
    Execute original action a_t, observe reward r_t and next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}_{\text{RL}}$ and $\mathcal{D}_{\text{AR}}$
  // AR phase
  repeat
    Sample mini-batch $\{s_t, a_t, r_t, s_{t+1}\}$ from $\mathcal{D}_{\text{AR}}$
    Update ϕ, ψ and ε using the sampled mini-batch
  until *reaching maximum AR training steps*;
  // RL phase
  repeat
    Sample mini-batch $\{s_t, a_t, r_t, s_{t+1}\}$ from $\mathcal{D}_{\text{RL}}$
    // latent action remapping
    $k_t = e_\psi(s_t, a_t)$
    Update agent networks using $\{s_t, sg[k_t], r_t, s_{t+1}\}$
  until *reaching maximum RL training steps*;
until *reaching maximum interacting environment steps*;

Network Architecture The Structure of our AD-VAE for hybrid action spaces is shown in 7. As discussed in 4.2.2, the input of the action encoder of AD-VAE includes the embedding of state and action. According to the inspiration of (He et al., 2016), we also add a skip connection of the state embedding to the action decoder. The output of the action decoder is divided into 2 heads to reconstruct the continuous part and the discrete part of the original hybrid action respectively. Specifically, for *Gym-Hybrid*, the original hybrid action is (action_type, action_arguments), where, the action_type is the discrete action that decides the type of the hybrid action, e.g. {Accelerate, Turn, Break}, the action_arguments is the two dimensional continuous action that decides the arguments of the hybrid action. We could use a graph neural network to model the relationship between the parts of the original hybrid action to get action_embedding. Without loss of generality, we simply take the one-hot encoding of action_type and concatenate it with action_arguments as our final action_embedding.

For the ensemble Q network, we utilize a common shared state encoder but have N ensemble Q-value heads, i.e., the penultimate layer of the Q network is connected to N linear layers and outputs N Q-value, then our Q update equation is shown in modified from Double DQN.

Hyper-Parameters In this subsection, we provide the hyper-parameters of our proposed ADQ method in We adopt the same hyper-parameters of the continuous action space baseline algorithm TD3, and the hybrid action space algorithm MPDQN, HPPO from their original paper.

Computational Cost All our experiments are performed on the NVIDIA V100 GPU. The experiments on *MuJoCo* environments with continuous action space taken approximately 8 hours to achieve training iterations on 3M env steps in each seed. The experiments on *Gym-Hybrid* environ-

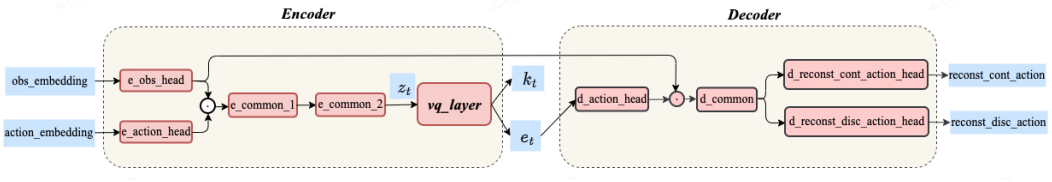


Figure 7: The Model Structure of our VD-VAE for hybrid action spaces. The input of the action encoder of AD-VAE includes the embedding of state and action. According to the inspiration of the paper, we also add a skip connection of the state embedding to the action decoder.

Hyper-parameter	Value
RL Hyper-parameter	Value
Discount factor	0.99
TD-step	3
Learning rate	3e-4
RL replay buffer size	1e6 (transitions)
AD-VAE replay buffer size	1e6 (transitions)
Hidden size List of Q network	[256, 256, 128]
Ensemble Number (N)	20
N sample per collect	256 (transitions)
Batch Size	512
Update per collect	50
AD-VAE Hyper-parameter	Value
Learning rate	3e-4
Hidden size List of encoder	[256, 256,256]
Batch Size	512
Size of embedding table (i.e. latent action shape)	128 (64 for *Hopper-v3*)
Dimension of embedding table	256
commitment loss weight β	0.25
reconstruct loss weight	10
Warmup data size	5e4
Warmup update steps	1e4

Table 1: Key Hyperparameters of ADQ used in (continuous action space) *MuJoCo* Environment.

ments with hybrid action space taken approximately 3 hours to achieve training iterations on 3M env steps in each seed. The experiments on *GoBigger* environments with hybrid action space taken approximately 8 hours to achieve training iterations on 3M env steps in each seed.

A.3 The Effect of Expert Data Warmup

In this section, we investigate the effect of different mechanisms of warmupin ADQ on *Hopper-v3* (continuous). The comparison curves are shown in Figure 8.

Concretely, we have the following two variants in total, and their brief descriptions are as follows:

Collection of Expert Data First, we train the TD3 agent until convergence, then use the best TD3 agent interact with the environment to collect 1000 episodes, then we only select the episodes whose return is larger than 3500 as the expert warmup dataset, in total, about 269800 transitions. When we pre-training AD-VAE, we set the epoches=20.

ADQ-expert-warmup: the ADQ variant agent pretrain the AD-VAE utilizing the expert data.

ADQ-expert-warmup w/o AR: the ADQ variant agent don't use *Continuous Action Regression* in AD-VAE, and AD-VAE is pretrained utilizing the expert data.

A.4 Different Mechanisms of Exploring the Original Action Space

There are two places in our NDRL framework that need to introduce exploration mechanisms, namely exploration in latent action space χ_{RL} and exploration in original action space χ_{AR}. Our

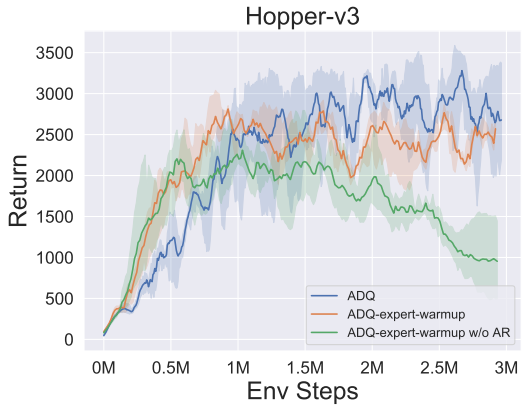


Figure 8: The Effect of different mechanisms of warmup in our VD-VAE. The x- and y-axis denote the environment steps ($\times 10^6$) and average episode return over 10 episodes, respectively. Curves and shading denote the mean and standard deviation over 3 seeds.

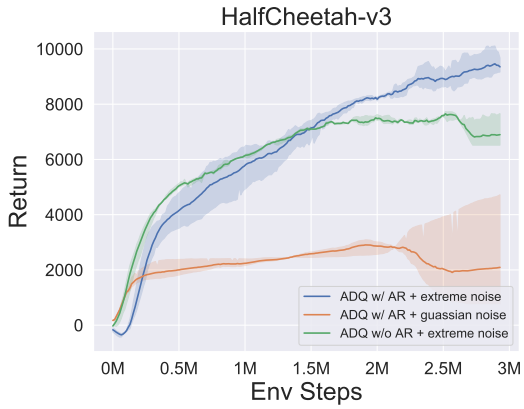


Figure 9: The Effect of different mechanisms of Exploring the Original Action Space in our VD-VAE. The x- and y-axis denote the environment steps ($\times 10^6$) and average episode return over 10 episodes, respectively. Curves and shading denote the mean and standard deviation over 3 seeds.

Hyper-parameter	Value
RL Hyper-parameter	Value
Discount factor	0.99
TD-step	3
Learning rate	3e-4
RL replay buffer size	1e6 (transitions)
AD-VAE replay buffer size	1e6 (transitions)
Hidden size List of Q network for *HardMove-v0-n10* and *GoBigger*	[256, 256, 128]
Hidden size list of Q network for *Moving-v0* and *Sliding-v0*	[128, 128, 64]
Ensemble Number (N)	20
N sample per collect	256 (transitions)
Batch Size	512
Update per collect	50
AD-VAE Hyper-parameter	Value
Learning rate	3e-4
Hidden size list of encoder	[256, 256, 256]
Batch Size	512
latent action shape for *Moving-v0* and *Sliding-v0*	16
latent action shape for *HardMove-v0-n10* and *GoBigger*	64
Dimension of embedding table	64
commitment loss weight β	0.25
reconstruct loss weight	10
Warmup data size	5e4
Warmup update steps	1e4

Table 2: Key Hyper-parameters of ADQ on (hybrid action space) *Gym-Hybrid* and *GoBigger* Environment.

instance method ADQ is essentially value-based, thus naturally, we adapt the usual epsilon-greedy exploration mechanism as χ_{RL}, i.e. $k_t = \epsilon - Greedy(Q(s_t, a_t))$. Another core problem is how to efficiently explore the original action space without affecting the stability of the NDRL framework.

Motivated by (Seyde et al., 2021b), we propose a special noise mechanism, namely, with a small probability (e.g. 0.1), we execute the random Bernoulli extreme action instead of the decoded original action. And we also experiment the usually Guassian noise mechanism proposed in (Fujimoto et al., 2018).

In this section, we investigate the effect of different mechanisms of exploring the original action space χ_{AR} on *HalfCheetah-v3* (continuous). The comparison curves are shown in Figure 9.

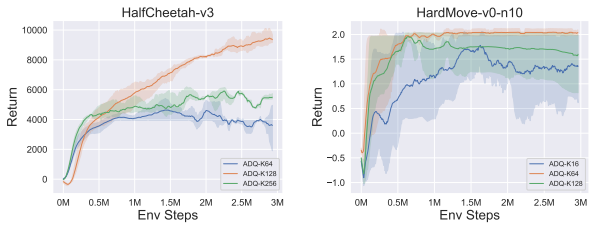
Concretely, we have the following three variants in total: **ADQ w/ AR + extreme noise**: the normal ADQ agent use *Continuous Action Regression* in AD-VAE, and when collecting data, we execute the random Bernoulli extreme action distribution with a small probability (e.g. 0.1).

ADQ w/ AR + guassian noise: the ADQ variant agent use *Continuous Action Regression* in AD-VAE, and when collecting data, we first add a Guassian noise into the decoded continous action same as in (Fujimoto et al., 2018), then use the noised action interacting with the environment. Specifically, the Guassian distribution is $\mathcal{N}(\mu, \sigma^2)$, and the clipped *noise_range = [-0.5, 0.5]*.

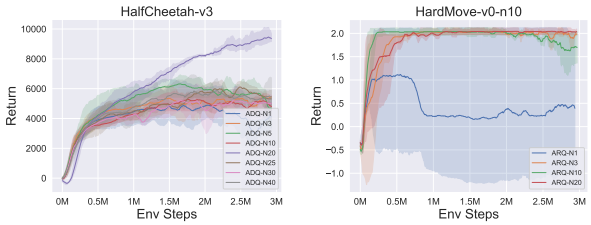
ADQ w/o AR + extreme noise: the ADQ variant agent don't use *Continuous Action Regression* in AD-VAE, and when collecting data, we execute the random Bernoulli extreme action distribution with a small probability (e.g. 0.1).

Figure 9 shows that when collecting data, if we add a guassian noise action into the decoded original continuous action, the performance of ADQ drops significantly. We conjecture that the reason for this is the normal action representation learning process is severely hindered by the continuous noise injection. And the performance of **ADQ w/o AR + extreme noise** is better than **ADQ w/ AR + extreme noise** verifying the effectiveness of *Continuous Action Regression* discussed in Section 4.2.2.

This results remind us that the exploration mechanism in the NDRL framework should be specially considered and designed.



(a) latent action shape K (b) latent action shape K



(c) ensemble head N (d) ensemble head N

Figure 10: The effect of latent action shape K and the number of head in ensemble Q network N. **Top**: The comparison curves for different latent action shape. **Bottom**: The comparison curves for different number of heads in ensemble Q network.

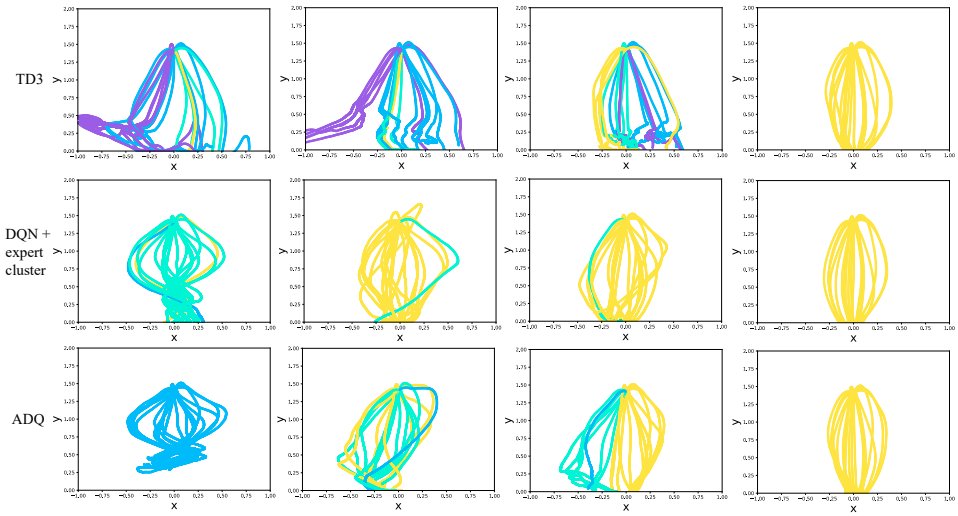
A.5 Experimental Analysis of Key Hyper-parameters

In our framework, there are two core hyper-parameters: the latent action shape K and the head number of ensemble Q network N, which largely determines the performance and stability of the algorithm.

In this section, we investigate the effect of these two key hyper-parameters in ADQ on *HalfCheetah-v3* (continuous) and *HardMove-v0-n10* (hybrid). The comparison curves are shown in Figure 10.

It seems that the head number of ensemble Q network N needs to be moderate for different environments. When N is too small, such as N=1, that is, when using the usual Q network, due to the over-estimation problem, the performance will diverge. When N increases, the stability of ADQ gradually increases. But N is not the bigger the better. When N increases to a certain value, it will become unstable due to the increased learning burden of the Q network. We found that for different environments, due to the difference in state shape and action shape, the best number of N often varies, e.g. 20 for *HalfCheetah-v3*, 10 for *HardMove-v0-n10* in our setting. The very similar phenomenon also occurs on the latent action shape. The best performing latent action shape is 128 for *HalfCheetah-v3*, 64 for *HardMove-v0-n10* in our setting.

A.6 Additional Visualization Results



(a) 0.1M Env Steps (b) 0.2M Env Steps (c) 0.5M Env Steps (d) Best Env Steps

(a) Episode Trajectory Distribution

Figure 11: Landing path of the spaceship during different training stages in 3 algorithms on the same *LunarLander* environment. Trajectories with color closer to red means higher episode return, i.e. yellow $>$ green $>$ blue $>$ purple, the range of whose episode return is $(-inf, -100), [-100, 0), [0, 200), [200, inf)$, respectively. TD3 (original continuous action space), DQN + expert cluster (discrete action space obtained by clustering on TD3 expert data), ADQ (discrete action learned by AD-VAE from scratch). **Bottom**: Landing path of the spaceship during different training stages. Darker trajectories means higher episode return.