Inferring DQN structure for high-dimensional continuous control

Andrey Sakryukin [1] **Chedy Raïssi** [2 3] **Mohan S. Kankanhalli** [1]

Abstract

Despite recent advancements in the field of Deep Reinforcement Learning, Deep Q-network (DQN) models still show lackluster performance on problems with high-dimensional action spaces. The problem is even more pronounced for cases with high-dimensional continuous action spaces due to combinatorial increase in the number of the outputs. Recent works approach the problem by dividing the network into multiple parallel or sequential (action) modules responsible for different discretized actions. However there are drawbacks to both the parallel and the sequential approaches, i.e. parallel module architectures lack coordination between action modules, leading to extra complexity in the task, while a sequential structure can result in the vanishing gradients problem and exploding parameter space. In this work we show that the compositional structure of the action modules has a significant impact on the model performance, we propose a novel approach to infer the network structure for DQN models operating with high-dimensional continuous actions. Our method is based on uncertainty estimation techniques and yields substantially higher scores for MuJoCo environments with high-dimensional continuous action spaces, as well as a realistic AAA sailing simulator game.

1. Introduction

Deep Reinforcement Learning (RL) has gained special attention from the research and practitioners community after demonstrating its ability to handle problems with high-dimensional input spaces (Mnih et al., 2015). Following that, a number of works applying RL in various areas, including Robotics (Polydoros & Nalpantidis, 2017), Computer Vision (Yun et al., 2017), NLP (Li et al., 2016) and others, were published. One of the main focuses of recent RL research has been tackling complex input spaces and/or compound problems. Off-policy models such as DQN are of special interest in the field, thanks to their sample efficiency, ease of implementation, and training time. However, in spite of its success with high-dimensional input spaces and complex tasks, increase in the number of actions brings a lot of challenges to their training process. High-dimensional action spaces lead to the problem of maximization of an arbitraty function over the set of possible actions. The problem is even more challenging for the case of high-dimensional continuous actions, requiring fine-grained control. In this case a set containing m actions to be discretized by b bins would result in an action space of size b^m.

Recent works, focusing on the problem of high-dimensional continuous action spaces, approach the problem by separating the network into multiple action specific sub-networks (denoted action modules) connected to a shared component, which results in a linear growth in the number of network outputs. Existing works, applying sequential (Metz et al., 2017) or parallel (Tavakoli et al., 2018) module composition, assume that the set of actions modules follows either (i) a total order or (ii) that their order has no significant impact on the network performance. In this work we show that agent actions sub-networks can be partially ordered and that a modular composition architecture indeed affects the model performance. The use of a modular architecture allows extra information to flow to specific sub-networks, while mitigating problems such as vanishing gradients (e.g, with a sequential architecture). While in some tasks, like human locomotion, the composition architecture might be intuitive (e.g, knees and ankles are more important than shoulders for a walking task), in others, like boat sailing, it might not be trivial (how are sail trimming, sail steering and boat steering actions related to a run downwind?).

To determine such an architecture, we propose a simple approach relying on the generic uncertainty estimation of action modules. The key insight is that the overall performance is improved by placing modules with high uncertainty lower in the network so that they can leverage extra information coming from more confident modules.

At a high level our algorithm uses one of two novel strategies

[*]Equal contribution [1]School of Computing, National University of Singapore [2]INRIA Nancy Grand Est, France [3]Ubisoft, Singapore. Correspondence to: Andrey Sakryukin <asakryukin@u.nus.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

that we developed for the estimation of the action module output uncertainty. After collecting enough statistics over a limited number of game runs, the algorithm outputs a partially ordered set of action modules. This part is done using a simple clustering algorithm over temporal data. Our approach can naturally be adapted to any algorithm from the DQN family. We illustrate this versatility by using Double Deep Q Network and NoisyNet DQN.

To summarize, the contribution of our work is as follows: (i) we propose two novel approaches for uncertainty estimation in DQN; (ii) we show that agent actions modules should be devised as a partially ordered set and that this modular composition affects drastically the model performances; (iii) we propose a simple generic strategy for modular composition inference from uncertainty values through a clustering algorithm; (iv) we achieve state-of-the-art performance on MuJoCo environments (Todorov et al., 2012) and a realistic but complex AAA game among other DQN methods operating on high-dimensional outputs.

Our code is available at: `https://github.com/asakryukin/InferringDQN`

2. Related Work

Our approach is mainly built on top of three areas: RL for high-dimensional action spaces, Compositional Modular Neural Networks and uncertainty estimation in RL.

Reinforcement learning for high-dimensional continuous action space. Initial works on high-dimensional output spaces approached the problem by learning low-dimensional continuous embeddings for large action representations. (Pazis & Parr, 2011) combined this strategy with a binary search while (Dulac-Arnold et al., 2015) studied the addition of k-nearest neighbours method on top of it. However, these on-policy approaches are not end-to-end trainable. Later, (Tampuu et al., 2017) proposed the use of independent agent for different sets of actions. Although this approach can be applied to off-policy models and is end-to-end trainable, it has major convergence problems (Matignon et al., 2012). Recent progress was achieved by focusing on modular architectures (Tavakoli et al., 2018; Metz et al., 2017). In these works, authors achieved state-of-the-art performances by allocating separate sub-networks (modules) for each agent's action. (Tavakoli et al., 2018) applied a parallel structure, where each action module simultaneously and independently makes predictions. Although, this model appears as an effective approach to handle high-dimensional output, it simply ignores dependencies among different action outputs. This fact is observable in more complex environments. Alternatively (Metz et al., 2017) applied an auto-regressive approach, where modules are connected sequentially and receive outputs from all previous sub-networks. This approach also has a number of drawbacks. Long sequence of actions results in the propagation of errors to lower levels of the network, vanishing gradients during the training phase and a constantly growing parameter space.

In this paper, we generalize these existing approaches by showing that agent actions modules can naturally be represented through a partially ordered set and that the modular structure has a significant impact on the model performance.

Compositional Modular Neural Networks. Early works on compositional Modular Networks focused on visual question answering task and designed different modules to be responsible for different tasks. (Andreas et al., 2016b; Hu et al., 2017) conditioned module composition on the type of the inputs and outputs. (Andreas et al., 2016a) later used syntactic analysis to infer module composition in NLP tasks. (Devin et al., 2017) applied compositional modular networks to RL agents. The modules were composed manually based on the task at hand. (Andreas et al., 2017) apply modular structure for on-policy methods, where each module is responsible for a pre-defined subtask. Their approach uses two networks trained simultaneously, where the first network learns sequential composition of sub-task modules and the second network learns sub-tasks. In this work we propose an approach to infer a compositional structure for off-policy RL agents. Our approach allows learning complex architectures beyond parallel and sequential structures.

Uncertainty estimation for Deep Q-Networks. One way to estimate uncertainty in predicted values is to use Bayesian DQN (BDQN) (Azizzadenesheli et al., 2018). The downside of BDQN is its high computational complexity requiring calculations of matrix decomposition at each run. Bootsrapped DQN (Osband et al., 2016) uses multiple randomly initialized network heads. The uncertainty is defined in this case as the standard deviation across predicted respective values. Alternatively (Clements et al., 2019) proposes using quantiles (multiple outputs for each output bin) with a novel loss function. The final uncertainty is then defined as a standard deviation across respective quantiles.

In this work we propose a novel approach to estimate uncertainty across DQN action modules instead of separate values. Our method is light-weight and does not require additional parameters.

3. Approach

3.1. Network

Double Deep Q Network. We base our work on an off-policy model introduced by (Van Hasselt et al., 2016). Double Deep Q Network (DDQN) has led to super-human performance on multiple RL environments including Atari games. We use DDQN because of its simplicity and relative stability of the training process. DDQN is also used as a basis for more sophisticated approaches, such as Dueling DQN (Wang et al., 2015), Rainbow DQN (Hess et al., 2016), etc, which makes our approach more amenable for generalization.

DDQN networks are trained with a backpropogation algorithm optimizing the following loss function:

$$\mathcal{L} = (Q(s_t, a_t, \theta_t) - [r_t + \gamma Q(s_{t+1}, Q(s_{t+1}, \arg\max Q(s_{t+1}, a, \theta_t), \theta_t'))])^2 \quad (1)$$

where θ are the network parameters of the main network, θ' are the network parameters of the target network, s_t is the input state at timestep t, a_t the action at timestep t and $Q(s, a, \theta)$ is the network output representing the expected Q-value.

Noisy DDQN. Noisy DDQN (NDDQN) (Fortunato et al., 2017) is a modification to the vanilla DDQN model allowing lightweight learning with stochastic network parameters. Learnable stochastic parameters are designed to drive agents exploration. We use Noisy DDQN for uncertainty estimation which leads to inferring the module composition structure. NDDQN are easy to implement and require only modification of fully-connected layers as following:

$$o = f(\mu^W + \sigma^W \odot \epsilon^W)i + \mu^b + \sigma^b \odot \epsilon^b \quad (2)$$

where f is the activation function, ϵ the uniform random variable, $\odot$ the element-wise multiplication and μ^W, σ^W, μ^b, σ^b are learnable parameters.

Our approach can be considered as an improvement and generalization of two existing techniques approaching the problem of high-dimensional action space, namely Parallel Action Branching DDQN (Tavakoli et al., 2018) and Sequential DDQN (Metz et al., 2017). In these works, authors address the problem of action outputs by allocating a separate sub-network (module) for each of the agent's continuous actions. All modules receive the state embedding from the main trunk network. Instead of learning parallel or sequential module compositions we aim at learning a more general structure.

Definitions. Let $\mathcal{C} = \{m_1, m_2, \dots, m_i\}$ denote the set of all actions modules in the network. We define the binary relation $\leq_\sigma$ over $\mathcal{C}$ as the *"confidence"* relation. When $m_1 \leq_\sigma m_2$, we say that module m_1 is more confident than m_2. $\leq_\sigma$ defines a partial order on $\mathcal{C}$ if it is reflexive, anti-symmetric, and transitive. In the next section, we define two possible binary relations $\leq_\sigma$ linking to uncertainty estimation between modules. In our approach, the main trunk is responsible for input encoding and its output is used as an input to every module. Formally,

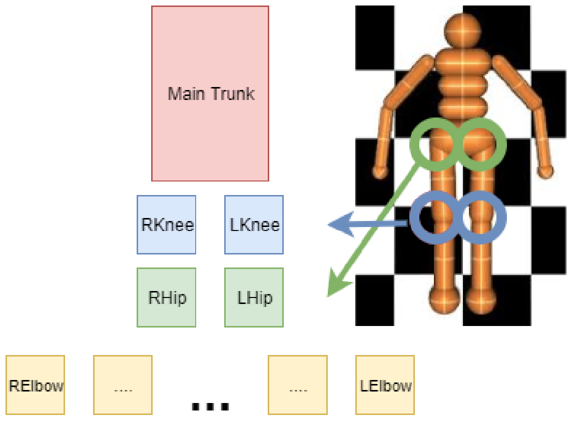

Figure 1. Schematic representation of our approach applied to Humanoid task.

$$\begin{cases} o_1^T = ReLU(s_t \times W_1^m + b_1^m) \\ o_l^T = ReLU(o_{l-1}^m \times W_l^m + b_l^m) \end{cases} \quad (3)$$

where o_l^T is the output of the l^{th} layer of the main trunk part. Action modules output Q-values. Each action module takes as an input the output from the main trunk concatenated with outputs of previous modules. Each module output is normalized for stability as the magnitude of outputs from other modules may be changing throughout the training process:

$$\begin{cases} o_{m_i}^A = & I_{m_i}^A \times W_{m_i}^A + b_{m_i}^A, \\ I_{m_i}^A = & o_f^T \bigoplus SoftMax(o_{m_k}^A - \min(o_{m_k}^A)\ ), \\ & \forall m_k \in \mathcal{C}\ s.t.\ m_k \leq_\sigma m_i \end{cases} \quad (4)$$

where $o_{m_i}^A$ is the output of action module m_i, o_f^T is the final output from the main trunk, $I_{m_i}^A$ is the input to action module m_i and $\bigoplus$ denotes the vector concatenation operation.

Similar to (Tavakoli et al., 2018) and (Metz et al., 2017) we define the total loss as $\mathcal{L} = \frac{1}{N}\sum_{m \in \mathcal{C}} \mathcal{L}_m$, where $\mathcal{L}_m$ is the loss function for action module m and $N = |\mathcal{C}|$. Since the assumption of our approach is that the less certain modules can benefit from leveraging the knowledge from more certain modules, we treat action module inputs I_m^A, $\forall m \in \mathcal{C}$ as constants and do not propagate gradients through them. An illustration of our approach is shown in Figure 1.

3.2. Uncertainty estimation

Our approach heavily relies on the uncertainty estimation throughout the learning process. To formalise our discussion on action module uncertainty we will rely on Bayesian modelling where two types of uncertainties are defined:

aleatoric and *epistemic*. Aleatoric uncertainty comes from the randomness in the environment itself and cannot be manipulated by the model. While epistemic uncertainty comes from the lack of agent's knowledge and is reduced by collecting more data. Due to the fact that different agent actuators have different effect on the reward and can be conditioned on different information, the epistemic uncertainty in different action modules decays with different trends throughout the learning process. We would expect that the most relevant actions would train faster, as their effect on the reward function is the highest, while others would lag behind and adjust to more certain ones. We assume that the learning process can be more efficient if modules with high epistemic uncertainty get extra inputs from more certain modules, thus providing more extra knowledge. Thus, the analysis of the initial module uncertainties and later grouping of corresponding modules into layers is at the core of our approach.

Noisy DQN. The problem of epistemic uncertainty in reinforcement learning is interleaved with the problem of efficient exploration. To reduce the epistemic uncertainty the network should get more training data. In reinforcement learning this translates to exploring more previously unseen states. Unlike vanilla DQN models that employ random exploration, Noisy DQN relies on two learnable sets of parameters: σ^W and σ^b representing the exploration rate. We denote the union of both sets of parameters as σ. As the agent gets more training, the values in σ^W and σ^b will tend to zero. In this work, for every action module, we will use the standard deviation over σ as an estimate of epistemic uncertainty.

Because of the presence of the stochastic component in equation 2, the average of σ will inevitably and monotonically converge to 0. Gradient clipping will also make the change in those values even closer. Thus, we focus only on the standard deviation values of σ for each of the action module. Indeed, modules with low standard deviation values over σ have lower uncertainty as they evenly minimize the mean value bounded by the gradient clip value. On the other hand, high standard deviation values signal that the module is keeping exploration high for some of the module parts.

As illustrated in Figure 2, we can see that the change is very uneven across parameters (Standard Deviation significantly out-weights mean). As illustrated in Figure 2, we can see that the distance between σ mean values is negligible compared to their standard deviation values, making any ordering of modules infeasible. While, as we can see from Figure 3 standard deviation across σ values are very consistent across different runs and directly affected by the change in module composition structure. Other uncertainty curves for other environments are provided in the Appendix.

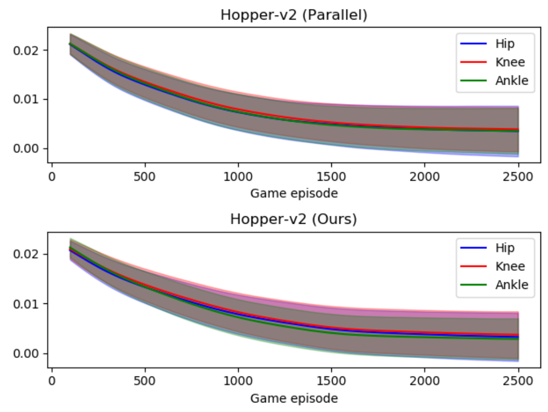

Figure 2. σ mean and standard deviation (shaded area) values across respective agent joints averaged across 5 games. (Top) indicated parallel composition of action modules, while (bottom) represents the structure inferred with our approach.

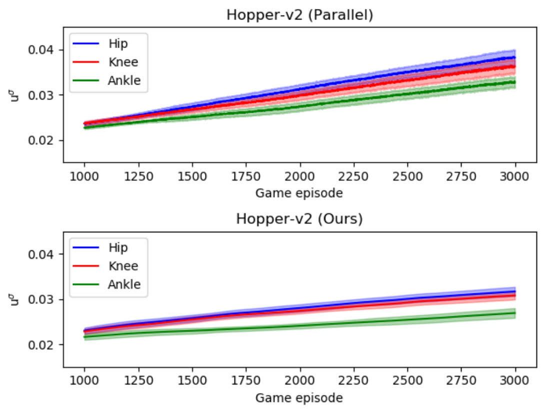

Figure 3. Average u^σ across 5 games for the Hopper environment.

Thus we define epistemic uncertainty as:

$$u_i^\sigma = f_\sigma\left(\bigcup_j (\sigma_{ij}^W \cup \sigma_{ij}^b)\right) \tag{5}$$

where u_i^σ is the uncertainty of the action module i, f_σ is the notation for the standard deviation of the values, i is the module index, j is the module layer index. An example of module uncertainties on the Hopper environment is shown in Figure 3. All the uncertainty estimations in our experiments are conducted with parallel structure, which doesn't assume any information exchange between modules. To capture the trends and dynamics in the uncertainty values we record these values as a time series.

DDQN Although, as will be shown in the experiments section, NDQN based uncertainty results in good performance, it requires more computations and longer execution time as it incorporates random components. Thus here we propose a strategy for uncertainty estimation in vanilla DDQN models.

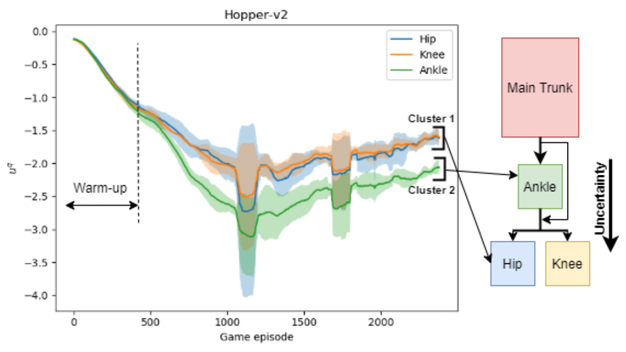

Figure 4. Schematic representation of the structure inference stage from u^q values for the Hopper environment. The resultant $c = \{\{\text{Ankle}\},\{\text{Hip,Knee}\}\}$

To estimate uncertainty level in such models we use statistics over model Q-values. Throughout the learning process the agent is trained to output maximum expected reward estimate. The final action is picked based on the maximum Q-value predicted. When the agent starts to learn, the output Q-values are of the same scale. As the agent becomes more accurate in its predictions, it starts outputting more diverse values, heavily prioritizing optimal actions. In other words, the more certain the model, the higher the standard deviation in its output values. Thus, we define (epistemic) **un**certainty as:

$$u_i^q = -\frac{\sum_t \sum_j (o^a_{itj} - \overline{o^a}_{it})^2}{t \cdot j}, \forall t, \forall j \tag{6}$$

where o^a_{itj} is the output from action module i at time-step t with index j, $\overline{o^a}_{it}$ is the mean value of the output vector o^a_{it}.

3.3. Structure Inference

While the importance of different actions and their respective composition may be intuitive in some tasks, it can be fairly obscure in others. To address this we propose a generic strategy for the inference of the compositional structure. To infer the module composition structure we want to group action modules with similar uncertainty dynamics together, while placing more certain modules higher in the hierarchy. In this case, actions with smaller impact on the reward function can benefit by conditioning their outputs on the information from more relevant ones.

We start by running a training with parallel architecture. During this stage we record uncertainty values for different action modules. As a next step we cluster module uncertainties into similar groups with k-means clustering algorithm and detect the optimal structure using elbow method (Kodinariya & Makwana, 2013) (i.e, the clustering technique allows us to partition our partially ordered set over $\mathcal{C}$ into k distinct sets on the same horizontal level). Since we record uncertainty values as a time series we use Dynamic-Time Warping (Berndt & Clifford, 1994) distance measure. The algorithm for the structure inference is presented in Algorithm 1.

Algorithm 1 Inferring Module Composition Structure

```
Function InferSctruct($n \leftarrow$ number of runs, $S$):
    all_us ← []
    for $i$=1; $i$<$n$; $i$++ do
        games_us← []
        while gamescore<$S$ do
            PlayStep()
            Train()
            if IsEpisodeFinished() then
                games_us.append(MeasureUncertainty())
            else
        end
        all_us.append(games_us)
    end
    $u$ ←mean(all_us,axis=0)
    $g$ ←cluster($u$)
    $c$ ←sort($g$)
    return $c$
```

We run stage 1 experiments for a limited number of iterations. As number of training iterations depends on the game on hand we define a stopping criteria for the structure inference stage (stage 1) as a score $S = k \cdot IS$, where IS is the initial model performance and k is the multiplier coefficient, which we further discuss in Section 4.2. For a more precise estimation we also average uncertainty values across n runs. Once we have our action modules clustered in N groups, we sort them based on average cluster uncertainty. Thus, clusters with higher uncertainty get located lower in the hierarchy. The rearrangement of the action modules usually does not change the ordering of the uncertainty curves, thus we do not need to reiterate the inference process. The schematic representation of the structure inference stage from u^q is present in Figure 4.

4. Experiments

First, we estimate the performance of the proposed method on a challenging task of continuous control in Mujoco (Todorov et al., 2012) Physics simulator. We compare performances on four environments: Hopper, Walker2d, HalfCheetah and Humanoid. We assign a separate action module for each joint with 17 discretized outputs, these setting showed best performance according to (Tavakoli et al., 2018). Complexity details for each environment can be found in Table 1.

Furthermore, we demonstrate that our approach is generalizable and applicable to environments without physically linked actions and non-intuitive relations. To do so, we

	N. of joints	action space
Hopper	3	4,913
Walker	6	$2.40*10^7$
HalfCheetah	6	$2.40*10^7$
Humanoid	17	$8.27*10^{20}$

Table 1. Details of the MuJoCo environments

apply our method to a AAA sailing game[1]. Agent in this environment has three actuators operating in continuous space and discretezed into 25 bins each. The actions are: sail rotation angle, steering wheel rotation angle, sail opening. The agent's task is to learn to navigate the ship to a destination while avoiding any obstacles in a complex physics environment. The environment takes into account alterating wind (direction and force), waves (strength), water depth, obstacles (shores and other agents). The agent gets a maximum reward of 1.0 for reaching the destination, travelled distance reward in the range from -0.01 to +0.01 and -0.01 reward for any collision.

Network details. In this work we used DDQN model with target network frequency update of 1000. The network main trunk is a 2 layer MLP with 512 and 256 neurons. Action modules were represented by a hidden layer of 128 neurons. All layers use ReLU activation function. The replay buffer size was set to 1 million entries.

4.1. Baselines

We start comparing the performance of our approach to the Independent DDQN (IDDQN) (Tampuu et al., 2017), where each actuator is learned by a separate agent and the problem is approached as a Multi-Agent environment. Next we compare to alternative approach using parallel (BDDQN) (Tavakoli et al., 2018) and sequential (SDDQN) (Metz et al., 2017) module composition. Finally, although the problem and the proposed solution are specific for the family of DQN algorithms, we evaluate the performance of Deep Deterministic Policy Gradient method (DDPG) an off-policy algorithm operating with continuous outputs.

For the case of AAA sailing game, due to game engine constraints, we compare performance only to BDDQN and SDDQN discrete methods.

4.2. Results and Analysis

First we start by analysing the performance of our approach to baselines. We run experiments in two settings: with composition derived from Q-value statistics and composition based on Noisy DDQN σ statistics. On the result plots these architectures referred as [Q] and [S] respectively, if the architecture of both approaches coincide we denote is as

[1]The game is not publicly announced at the submission time.

	Hopper	Walker 2d	Half Cheetah	Huma- noid	AAA
DDPG	**3595**	4349	5871	2251	-
IDDQN	3185	4328	8371	3119	-
BDDQN	3374	3830	9026	3990	1.6
SDDQN	3415	3860	8979	3233	1.3
Ours [Q]	3495	4552	**9222**	**4224**	**2.2**
Ours [S]	3495	**4855**	**9222**	4074	**2.2**

Table 2. Maximum total rewards averaged across 30 games.

[S|Q]. In Figures 5 - 6 we present smoothed (window size of 20 episodes) learning curves averaged across 5 runs with standard deviations (shaded areas). As it can be seen, our approach outperforms all discrete space baselines. As the environment complexity increases the improvement margin of our approach is getting more significant. Also, starting from Walker2d environment our approach does outperform DDPG algorithm operating with continuous actions. Moreover, as we can see from Table 2 both our approaches consistently outperform other discrete space techniques in terms of maximum average cumulative reward per game. Moreover, as it can be seen from Figure 7, our approach achieves significant improvement applied to the AAA sailing game.

Module composition. Next we evaluate the effect of the compositional structure on the model performance and consequently show that set of actions is partially ordered. In Figures 8 - 9 we compare learning curves with architectures derived from Q-value statistics, σ values, 'real' agent skeleton structure and inversed structures. We refer to 'real' composition as the one translated from the physical structure of the agent. Although it is intuitive for the MuJoCo environment, it is not always applicable like in the case of a sailing agent. An example of a 'real' structure for the Walker2d environment is c={{LHip,RHip},{LKnee,RKnee},{LAnkle,RAnkle}}. Further details on the used structures can be found in the Appendix. As it can be seen, the average model performance is indeed significantly impacted by the network architecture. Importantly Q-value and σ based approach result in similar performances.

As presented in Table 5 both of introduced uncertainty estimation techniques result in similar compositional structures. The main differences between them are grouping of neighbouring clusters and/or swaps in the elements of neighbouring clusters. Furthermore, in case of MuJoCo, the inferred structures are also close to the 'real' structures in terms of composition, as well as performance. Notably, both approaches have placeed most relevant actions on top of the hierarchy. In the case of the one legged Hopper environment, the ankle joint was located at the top of the hierarchy, while in Walker2d and HalfCheetah environments, where the agent balance depends on two limbs, the preference was

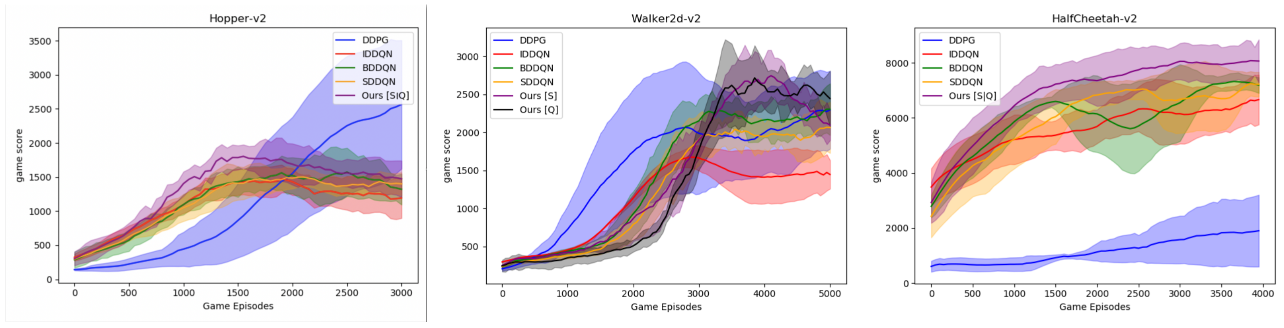

Figure 5. Comparison of model performances. Solid lines represent smoothed (window 20) game reward averaged across 5 runs. *y* axis represents total game score, *x* axis represent number of training episodes, shaded areas represent standard deviation. Model performance was evaluated every 50 training episodes and averaged for 30 games.

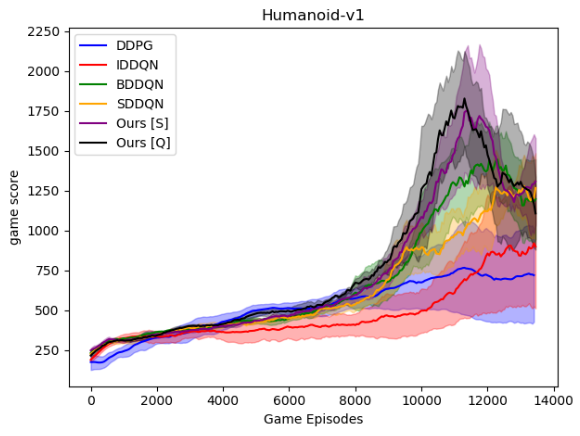

Figure 6. Comparison of model performances.

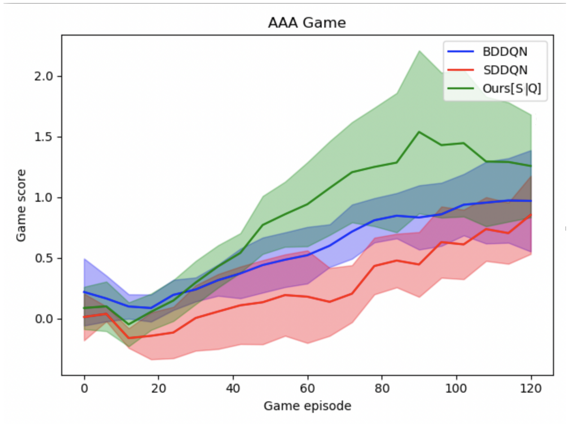

Figure 7. Comparison of model performances.

given to Hips. Similarly, in Humanoid case, leg actuators were placed above the rest.

Ablation study. In this section we demonstrate the effect of the hyper-parameters on structure inference stage. First we estimate the effect of the number of training episodes

	k			
	1.0	1.25	1.5	2.0
Hopper	✓	✓	✓	✓
Walker	X	X	✓	✓
HalfCheetah	X	✓	✓	✓
Humanoid	X	X	✓	✓

Table 3. Success to infer structure after reaching k*IS score identical to full training based structure. Results evaluated across 5 runs.

on the accuracy of the inferred structure. Here, we define the performance as successful if inferred structure matches the one inferred from the complete training cycle averaged across 5 games in more than 80% (4 our of 5) of trials. Because different games require different number of training iterations, we define the initial score (IS) as an average total score achieved after a short warm-up. Here we measure IS after 350 games (15k iterations) to allow to populate the replay buffer and start model improvement. We use it as a reference for the length of uncertainty estimation cycle, for example, if we run uncertainty estimation till $2.0 \cdot IS$ ($k = 2.0$) score, this means we stop the training cycle with parallel structure after the model reaches the score of $2.0*$IS and start structure inference. The corresponding results of the ablation study can be seen in Table 3.

Next we estimate the number of runs n required to get a stable structure result. To do so we run the structure inferring experiments with different number of runs. The results are presented in Table 4

As expected longer stage 1 with more averaged runs result in a more precise structure inference. Also, it can be seen that more complex environments require longer estimates (also requiring longer training time). However, as we can see even for complex environments as Humanoid having 17 action modules suffice 3 average runs with 1.5*IS stopping threshold.

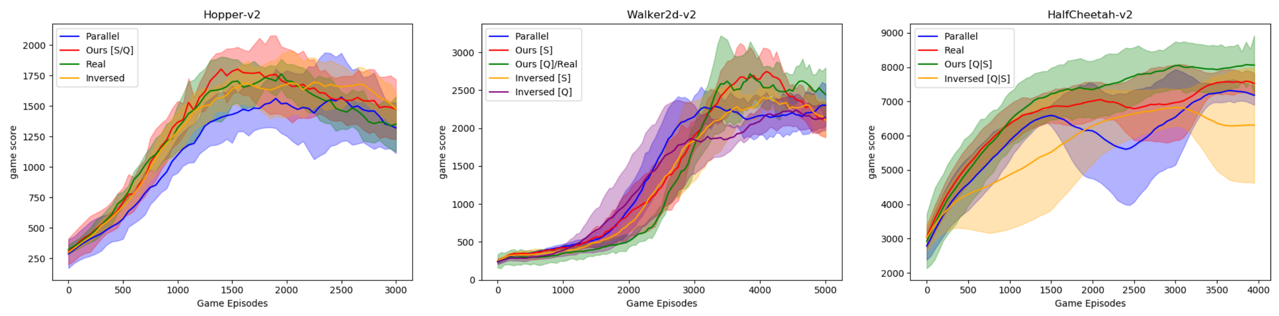

Figure 8. Comparison of action module composition architectures. Solid lines represent smoothed (window 20) game reward averaged across 5 runs. y axis represents total game score, x axis represent number of training episodes, shaded areas represent standard deviation. Model performance was evaluated every 50 training training episodes and averaged for 30 games.

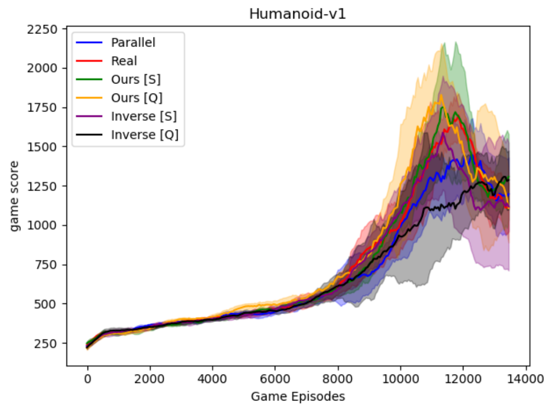

Figure 9. Comparison of action module composition architectures.

	n				
	1	2	3	4	5
Hopper	✓	✓	✓	✓	✓
Walker	X	X	✓	✓	✓
HalfCheetah	X	✓	✓	✓	✓
Humanoid	X	X	✓	✓	✓

Table 4. Success to infer structure using average of n column value), with k fixed to 1.5. Results evaluated across 5 runs.

5. Conclusion

In this work we have proposed a novel approach to reinforcement learning problems with high-dimensional continuous action spaces. We have introduced two uncertainty estimation approaches (σ and Q-values based), and uncertainty based strategy for compositional structure inference. Our method demonstrated state-of-the-art performance among other discrete space methods, as well as significant improvement compared to continuous space approach (DDPG) on tasks with higher complexity. Importantly, our approach applicable to generic cases where the composition is not intuitive. As a future work, we continue working on investigating end-to-end trainable approach for structure inference.

	u^σ	u^q
Hopper	{{Ankle}, {Knee,Hip}}	{{Ankle}, {Knee,Hip}}
Walker2d	{{LHip},{RHip}, {LKnee,RAnkle}, {LAnkle,RKnee}}	{{LHip,RHip}, {LKnee,RKnee}, {LAnkle,RAnkle}}
Cheetah	{{FHip,BHip}, {FKnee,BKnee, FAnkle,BAnkle}}	{{FHip,BHip}, {FKnee,BKnee, FAnkle,BAnkle}}
Humanoid	{{RKnee,LKnee}, {RHip$_y$,LHip$_y$}, {RHip$_z$,LHip$_z$}, {Abd$_x$,Abd$_y$, Abd$_z$, RHip$_x$, RShldr1, RShldr2, LHip$_x$, LShldr1, LShldr2 RElbow, LElbow}}	{{RKnee,LKnee}, {RHip$_y$,LHip$_y$}, {Abd$_x$,Abd$_y$, Abd$_z$, RHip$_x$, RHip$_z$, RShldr2, LHip$_z$, RShldr1, LHip$_x$, LShldr1, LShldr2, RElbow, LElbow}}

Table 5. Module composition structures obtained as a result of clustering uncertainty estimations based on u^σ and u^q.

Acknowledgments

The authors would like to thank Prof. Yarin Gal for the interesting discussions on the uncertainty estimations and the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

References

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Learning to compose neural networks for question answering. *arXiv preprint arXiv:1601.01705*, 2016a.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016b.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. Efficient exploration through bayesian deep q-networks. In *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9. IEEE, 2018.

Berndt, D. J. and Clifford, J. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pp. 359–370. Seattle, WA, 1994.

Clements, W. R., Robaglia, B.-M., Van Delft, B., Slaoui, R. B., and Toth, S. Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*, 2019.

Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2169–2176. IEEE, 2017.

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

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Hu, R., Andreas, J., Rohrbach, M., Darrell, T., and Saenko, K. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 804–813, 2017.

Kodinariya, T. M. and Makwana, P. R. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Matignon, L., Laurent, G. J., and Le Fort-Piat, N. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

Pazis, J. and Parr, R. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1185–1192, 2011.

Polydoros, A. S. and Nalpantidis, L. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

Tavakoli, A., Pardo, F., and Kormushev, P. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Yun, S., Choi, J., Yoo, Y., Yun, K., and Young Choi, J. Action-decision networks for visual tracking with deep reinforcement learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2711–2720, 2017.