Understanding Structure of Concurrent Actions

Perusha Moodley[1][0000−0001−6227−9578], Benjamin Rosman[2,3][0000−0002−0284−4114], and Xia Hong[1][0000−0002−6832−2298]

[1] University of Reading, Reading, UK
perusha.moodley@pgr.reading.ac.uk
[2] University of the Witwatersrand, Johannesburg, South Africa
[3] CSIR, Johannesburg, South Africa

Abstract. Whereas most work in reinforcement learning (RL) ignores the structure or relationships between actions, in this paper we show that exploiting structure in the action space can improve sample efficiency during exploration. To show this we focus on concurrent action spaces where the RL agent selects multiple actions per timestep. Concurrent action spaces are challenging to learn in especially if the number of actions is large as this can lead to a combinatorial explosion of the action space.
This paper proposes two methods: a first approach uses implicit structure to perform high-level action elimination using task-invariant actions; a second approach looks for more explicit structure in the form of action clusters. Both methods are context-free, focusing only on an analysis of the action space and show a significant improvement in policy convergence times.

Keywords: Concurrent actions · Reinforcement learning · Structure · Action elimination · Clustering

1 Introduction

In reinforcement learning (RL) complex environments are often modelled using games such as Atari [1], Go[13] and StarCraft[16] as proxies for learning how to behave in the real world. Many of these games can handle multiple actions per time step, but most algorithms unroll the possible combinations of actions and treat each combination as an individual action (or primitive action) during learning [17, 8]. This approach has been successful in many complex environments but as environments become more specialised, integrated and relevant to the real world it becomes increasingly important to learn something of the structure of the action space to improve exploration times and leverage relationships or causal patterns.

This paper looks specifically at environments where the RL agent can take multiple simultaneous discrete actions in a timestep, also referred to as concurrent action environments. These environments may apply the actions concurrently or sequentially but will return a single response to the concurrent action request from the agent, for example, action set A = (jump, kick) elicits a response

(next-state=position-x, reward=0.01) from the environment. Rohanimanesh et al. [10] define concurrent actions and a generalised framework for working in such environments. The action space consists of a pool of primitive actions from which the agent must select one or more actions to send to the environment. There is potential for interactions, relationships and influence between actions, for example jumping and kicking vs just kicking may trigger better rewards.

Concurrent action settings are becoming more relevant as RL environments become more complex with actions that interact and may be applied concurrently, for example Atari [1] and StarCraft II [16] environments. Concurrent actions are possible in a single agent and a multi-agent reinforcement learning (MARL) context. While exploration in a concurrent action MARL environment [4, 3] is an interesting area it is out of scope in this paper.

A concurrent action environment is complex because the computation time required to explore all state-action-sets increases combinatorially with the number of actions [14]. It is desirable to reduce the amount of exploration required, either by using prior knowledge or some intelligence about the environment. An important aspect of these environments is the possibility of relations and interactions between actions. Treating action combinations as primitive actions ignores potentially valuable information, such as whether some actions work particularly well with other actions, or whether some combinations of actions are redundant.

Some work in concurrent action spaces has focused on addressing these problems by reducing the size of the space using action elimination and prior knowledge [11, 18]. Other groups have looked at various representations of the action space to reveal relationships and structure between actions either by factorisation, encoding or embeddings [17, 12, 15, 2].

The scope of this work is the single agent concurrent action environment where the agent takes multiple discrete actions per timestep, or an action set. In this paper, we propose extracting structure from the action space to better inform exploration and improve sample efficiency. Two mechanisms are compared: explicit structure in the form of clusters in action space and implicit structure exposed by task-invariant actions. Both are unsupervised approaches focused purely on the action space, i.e. context-less.

Frequently used actions are collected from successful trajectories in a pre-training phase. The frequency of the action set is used as an indicator of how useful it is for a task. The most frequent action sets across multiple tasks are collected, representing a set of task-invariant action sets, and used as a prior during exploration in the second phase of training. The task invariant action sets indicate some underlying structure that action elimination can expose and leverage.

The frequency data is also used to learn more explicit structural information using spectral clustering [6] to partition the action space and reveal clusters of actions that work well together.

The contributions in this work are:
1. Using action elimination to reduce the exploration space in a concurrent action environment (Section 4).

2. Finding explicit structure in the concurrent action space using spectral clustering (Section 5).

These approaches were inspired by observations of how actions interact and influence other actions in the real world and how we, as humans, learn to select actions. We have a distinct advantage because we have a vast amount of preprocessed and distilled experience. When we need to learn something new we are very reliant on existing abstractions and structure. We also always apply relevance filters, often but not always, filtering by context. We are able to apply actions/skills learnt in one context to a completely new context without ever having seen the new context. This implies an abstraction of the mechanism in the action itself.

Section 2 looks at related research. Section 3 reviews the reinforcement learning framework and how it is used in the concurrent action problem setting. Section 4 expands the action elimination algorithm proposed to reduce exploration space while Section 5 looks at explicit structure as a means to enhance exploration. Section 6 provides details of the experiments and results. Finally, Section 7 concludes and discusses future work.

2 Related Work

Several papers combine learning factors for action composition in concurrent environments although the approaches for obtaining the factors and methods of composition differ. Wang and Yu [17] decompose an action into sub-action components and focus on learning the relationships between sub-actions. They use a novel structure to model the parameters of the sub-action components in a maximum a posteriori setting to learn the relations between sub-actions. Similarly Sharma et al [12] decompose every action into a set of action-factors forming a factored action representation. The agent learns how to compose concurrent sets of actions based on the factors. Harmer et al. [5] also adopt the composition approach and propose a network architecture that outputs multiple actions per timestep in a deep RL setting. The network is trained using auxiliary signals from experts resulting in an online approach, one of the few online models. The agent benefits from having a concurrent action structure as it can model and learn from experts without restriction. This approach seems very effective but requires expert data versus self learning.

Rosman et al. [11] and Zahavy et al. [18] look at action elimination and prior knowledge to bias the agent's learning of new behaviours. In Rosman et al. action priors are modelled using Dirichlet distributions where the concentration parameters are the counts for a task. Zahavy et al. propose reducing the size of the relevant action set per state by using a separate network, a bandit trained using an elimination signal, to control which actions to eliminate in the main RL Deep Q-Network agent. The motivation is to remove unnecessary actions and improve sample efficiency.

Some interesting work that focuses specifically on structure in action space comes from Tennenholtz et al. [15]. They developed an action-context embed-

ding representation, Act2Vec, modelled on word embeddings using skip-grams [7]. The action embedding is used to enhance Q-function learning and to cluster similar actions together, thereby reducing the exploration space. The embeddings are generated from optimal demonstrator trajectory data. The paper uses this representation and a new measure of similarity to consider a broad spectrum of analysis that includes concurrent actions and exploration across clusters. Chandak et al. [2] also learn a lower dimensional action representation and a transform function between the representation policy and original policy. The embedding is used for training the agent in the lower dimensional space and makes use of underlying structure, similar to Tennenholtz et al.

Our paper has expanded on some of the ideas in Rosman et al. [11] but in a concurrent action setting specifically extracting task-invariant structure into a prior for action exploration. There are a few similarities in approach with Tennenholtz et al. They use supervised embedding and clustering of action space vs our unsupervised, count-based spectral clustering approach; we both look at the action-only context; the use of the clusters during exploration is also different. Tennenholtz et al. cluster to prevent redundant selections of actions whereas we cluster effective combinations with positive interactions.

3 Preliminaries

This work considers the setting of a standard Markov Decision Process (MDP) [14], defined by (S, A, T, R, γ) where S refers to the set of states, A the set of n primitive actions defined by $\{a_0, a_1, ..., a_n\}$, T the transition function $T : S \times A \times S \to [0, 1]$ defines the probability of moving from a state s to the next state s' after taking action a, R the reward function $R : S \times A \to \mathbb{R}$ such that $R(s, a)$ is the expected reward received when taking action a at state s and $\gamma \in [0, 1]$ is the discount factor.

The focus of this paper is concurrent actions as opposed to primitive actions. A concurrent action is represented as a set of actions $A_i \subset A$ containing np primitive actions. In this paper we focus primarily on the case where $np = 2$, so the agent takes two actions (hereafter referred to as an action set) per timestep $\{a_{i0}, a_{i1}\}$ where $a_{i0}, a_{i1} \in A$.

In the normal RL framework a stochastic policy $\pi : S \times A \to [0, 1]$ is defined where $\pi(s, a)$ is the probability of selecting an action a in state s and $Q^\pi(s, a)$ holds the value of each state-action pair under the policy π.

In the concurrent action environment the Q-value function holds the values of the action set, A_i, at each state. The goal is to find the optimal action set A_i that yields the highest value at any state s. While there is no restriction on how actions are selected, the actions are evaluated at the action set level.

Action selection generally takes place in the explore/exploit phase where the agent's accumulated knowledge of the environment is utilised or it is forced to explore new state-action-set combinations. Exploration can take a long time in a concurrent action space with a large number of actions. The next section

expands the proposed ideas of learning structure in the action space to improve the sample efficiency of the exploration process.

4 Implicit structure: Action Elimination using task invariance

This approach collects samples from successful trajectories across multiple tasks to extract or filter the most frequently used action sets.

The intuition is that high frequency action sets are probably more useful and could be used as a basis for elimination. To reduce bias a pre-training process is performed over multiple tasks to extract a set of task invariant action sets that are referenced during exploration. The motivation is to remove useless action sets by favouring the task-invariant sets thereby reducing the size of the exploration space and making exploration more efficient.

Algorithm 1: Processing Count Data

1. Generate $Counts$ **matrix**
Generate trajectories using RL algorithm (eg. Q-learning)
Collect count of each action set selected per state for successful episodes
Average over multiple runs to generate a count matrix by state and action set, $Counts(s, a, a)$

2. Process Action Counts
Hyperparameters: threshold t, no. of top counts NC
Reshape $Counts$ from SxNxN to SxN^2, unrolling action sets per state

2.a. Process Action Elimination Count Matrix
Set threshold $t > 0$ and NC to low number
Apply threshold filter to $Counts$ and accumulate only the remaining top NC action counts across all states
Normalise the vector formed: $probs(A_i)$

2.b. Process Clustering Count Matrix
Set threshold $t = 0$ and increase NC
Apply threshold filter to $Counts$ and accumulate the top NC action counts across all states
Reshape to form matrix W with dimensions N x N
Return Vector $probs(A_i)$ with dim 1xN^2 or in matrix form, W, with dim N x N, where N is no. of primitive actions

To collect the common or frequent action sets, a configurable environment is needed to easily create multiple tasks. Algorithm 1 describes how the count data is processed for both the action elimination method of this section (step 2a) and the clustering method of the next section (step 2b).

In a pre-training phase, a Q-learning algorithm is used to generate trajectories. The Q-function holds the value of each state-action set pair, $Q(s, A_i)$ where

A_i is the set of two actions $\{a_{i0}, a_{i1}\}$. Successful trajectories are collected where the goal state is reached before the maximum allowed step count (a hyperparameter).

The algorithm describes how the counts per action set are collected for a task. Only the most frequently used action sets are retained, controlled by a threshold hyperparameter, t. Counts below this threshold will be filtered out, effectively pruning the action sets. The result is a distilled action set usage vector representing each task. Note this approach holds count data by action set and not by action primitives.

Algorithm 2: Exploration with Action Elimination Prior

Given: $probs(A_i)$, $Q(s, A_i)$

If Explore:
    Sample an action set from $probs(A_i) \rightarrow A_i$
Else if Exploit:
    Find action set with max value in $Q(s, A_i) \rightarrow A_i$
Return A_i

This process is repeated for multiple tasks to build up a collection of invariant action sets for this domain. The action set counts are averaged across all tasks to produce a vector of importance weightings for each action set, which is then injected into the exploration process as prior knowledge. Algorithm 2 briefly illustrates how the action elimination prior is used to impact action selection in a basic Q-learning algorithm.

Averaging the counts over multiple tasks makes this a context-free approach, i.e. there is no direct association with states at this time. Experiments show that the training time is improved with this general context-free reduction of the action space.

The next section looks at an approach that finds and exploits explicit structure in concurrent action spaces.

5 Explicit Structure: Clustering of action space

The previous approach treats action sets as primitives, i.e. expands the action space to include all possible combinations of actions, then learns the best policy using a standard Q-learning algorithm. The general reduction in the number of eligible action sets is effective at improving time to convergence but the approach does not explicitly learn about the relationship between actions and is potentially discarding valuable information.

This second approach learns explicit structure in the action space in the form of clusters. The premise is that high counts of some action sets imply an underlying relationship or affinity between the primitive actions in those sets,

that can be used to separate the action space into clusters. During exploration and action selection the agent would select from within these clusters on the basis that there is a higher likelihood of picking an effective action set.

Algorithm 3: Spectral Clustering of Actions

Requires:
Count matrix for action sets, W, with dim N x N, where N is no. of primitive actions

1. Prepare Affinity Matrix:
Check Symmetry, zero diagonals

2. Apply Spectral Clustering algorithm [9]
Degree matrix: D where d_{ii} is the diagonal sum of row $W[i]$
Normalised Laplacian: $L = D^{-1/2}WD^{-1/2}$
V is a matrix formed from the top k of eigenvectors of L
Y is the matrix V normalised
Apply k-means to Y for k clusters and create list of clusters, C

3. Post Process Clusters
Remove clusters with low silhouette score
Remove single item sets
Return List of clusters C

Algorithm 3 describes how the spectral clustering algorithm in Ng et al. [9] is applied to cluster actions using a count matrix.

As in the previous approach frequent action sets are collected from successful trajectories using a Q-learning algorithm in a pre-training phase. Algorithm 1, step 2b describes the generation of the count matrix from trajectory data. The threshold and top counts hyperparameters are used to control the sparsity of the matrix.

As before this process is repeated across multiple tasks. An assumption of this work is that the partitioning of the action space does not change with the task however this will be considered in future work. A count matrix of dimension $N \times N$ is passed as input to Algorithm 3.

Spectral clustering is a clustering method that separates data using the eigenvectors of the Laplacian of an affinity matrix [6]. The affinity matrix should reflect the similarity (or affinity) of the component elements. The count matrix is used as the basis for an affinity matrix and spectral clustering is performed to find clusters in the action space. The count matrix may be viewed as a graph where each action is a node and the edge is the strength of the relationship between actions, the strength reflected by the count.

The spectral clustering algorithm applied [9] describes some conditions that the affinity matrix should meet. The first step in Algorithm 3 transforms the count matrix into the appropriate form. In step 2 the eigenvalues and eigenvectors of the normalised Laplacian are calculated; typically cluster blocks are

revealed in this step. K-means clustering is used to group the top eigenvectors. The number of clusters k is a hyperparameter, however it is also determinable using an eigengap heuristic [9]. A final post processing step uses a cluster measure to check the validity of each cluster and retains only the most confident clusters.

Spectral clustering was applied on this data although another clustering method could be used instead. Spectral clustering works well on correlated data and the count matrix forms a natural affinity matrix.

Once the clusters are determined, this structural information is built into the action selection method of the exploration process. Algorithm 4 briefly shows how the clusters are used during exploration to compose an action set.

Algorithm 4: Action Selection - Clustering

Given: List of clusters C, $Q(s, A_i)$, no. of actions in a set np
If Exploration:
    Sample a cluster c_i from C randomly
    Select np actions from within $c_i \rightarrow A_i$
Else if Exploitation:
    Find action set with max value in $Q(s, A_i) \rightarrow A_i$
Return A_i

6 Experiments

Experiment Setup: A basic four room grid environment was used, Figure 1. The key requirement was an easily configurable environment for generating multiple tasks. In this environment the doors, start state (S) and goal state (G) are easily changed to create different tasks. A single goal state (G) is located in one of the rooms for each task. The set of primitive actions is fixed across all tasks: 6 actions viz. {U-Up, D-Down, L-Left, R-Right, O-Open, E-Enter}. There is a per-step penalty of -0.01 and the goal state has a reward of 10. There are no rewards associated with the doors. For each room configuration the agent was run for 50 runs of 100 episodes; each episode was truncated after 100 steps. Several different room configurations were generated to mimic different tasks using the same set of actions.

Agent: The agent is configured to take 2 actions per timestep. There are no limitations on the composition of the action sets so useless action sets such as {U,D} are possible. A basic Q-learning algorithm was implemented. It should be noted that other algorithms such as SARSA or Policy Gradients [14] would work too as both methods are algorithm agnostic. In the data collection pre-training phase the agent was designed to select an action set of two actions per step by either exploring or exploiting. Exploration involves selecting two primitive

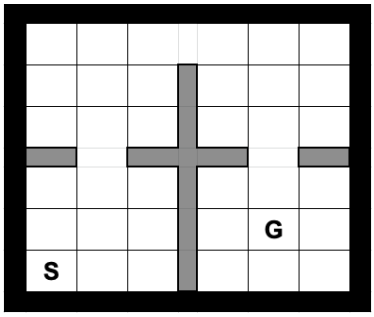

Fig. 1: Four room grid world

actions randomly with probability ϵ to create an action set. ϵ starts high at 0.9 and was annealed to 0. Exploitation is unchanged and performs a lookup of the maximum value action set in the Q-value function for any state.

6.1 Implicit Structure - Action Elimination

Eight tasks were chosen at random, each with a different room configuration. The agent was pre-trained using the Q-learning algorithm and counts of action sets by state were collected over the total runs and averaged. The counts were aggregated across all states for each action set and normalised resulting in a vector of proportional representation for all action sets. The threshold hyper-parameter was set to 7 and the number of top actions to 2 to generate a very sparse count matrix.

A sample of the resulting vector of proportions for the action sets is illustrated as heatmap in Figure 2. Each location on the grid is an action set, for example the first block on the top left is the action set (Up, Up). The colour intensity reflects how often that action set has been used in successful trajectories. The (Open, Enter) action set is relatively frequent, as is the (Down, Down) action set. The heatmap shows some of the bias that creeps in with only 8 tasks. Increasing the number of tasks during the first phase of training should help to reduce this.

The vector of proportions is injected into the exploration process as prior knowledge in the same domain but for a completely different set of tasks. Figure 3 compares the time to convergence for four randomly selected new tasks before and after applying action elimination. The graphs show a reduction in average steps to convergence implying that basic action elimination during exploration is effective in concurrent action spaces.

The results show that using the frequency of actions selected across tasks as a basis for eliminating less useful action sets, without any contextual reference to the state, is sufficient to result in a decent performance improvement.

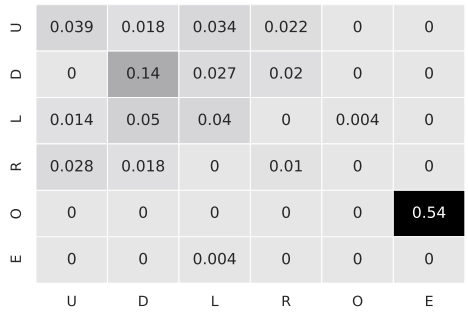

Fig. 2: Sample of a heatmap for action sets

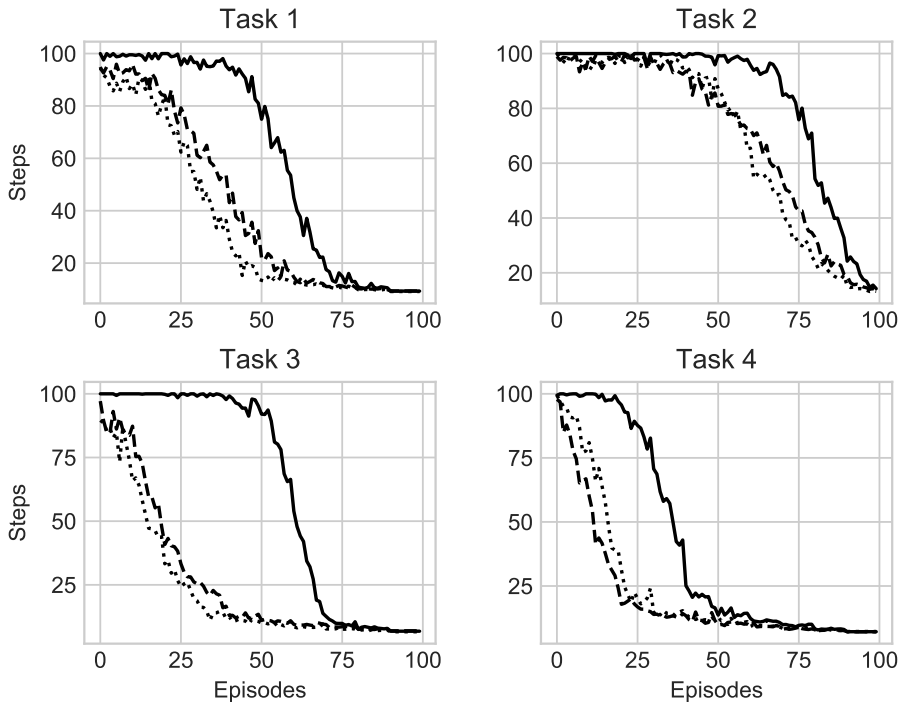

Fig. 3: Comparison of task convergence for pre-training(solid), clustering approach (dots), action elimination (dashed) for 4 random tasks

6.2 Explicit Structure - Clustering

The purpose of these experiments is to learn some structural aspects from the domain's action space that will improve the convergence times of other tasks in the same domain. To better illustrate this, the environment is designed with natural clusters in the action space. In particular a very specific set of actions was required to move through the doors in the four rooms environment, viz. (O, E) in a single timestep. Once again there were two phases: phase one was data collection pre-training phase across multiple tasks using a Q-learning algorithm with $\epsilon-$greedy exploration; phase two was training under the new exploration model (Algorithm 4).

The agent should learn that the O-Open and E-Enter actions form a cluster leaving the navigation actions $\{U, D, L, R\}$ to form a second cluster. In the second phase, action selection during exploration was constrained to intra-cluster selections rather than inter-clusters. This means the agent was selecting from door-related actions or navigation actions, which makes intuitive sense.

As mentioned above spectral clustering was applied to the count matrix to reveal a partition in the action space. The threshold was set to zero so no action sets were removed from the action space. The eigenvalues generated by the algorithm give an indication of how the matrix may be partitioned, particularly the second smallest eigenvalue. Plotting the eigenvectors of the first and second smallest non-zero eigenvalues shows a clear separation of the two sets of actions that corresponded to the door-related and navigation related groups expected (Figure 4).

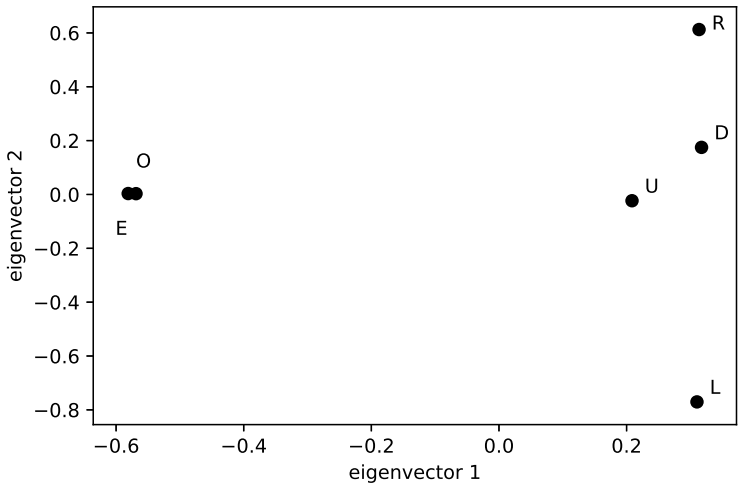

Fig. 4: Plotting the top eigenvectors shows separation of the Open, Enter actions and the navigation actions Up, Down, Left, Right

The action selection method in Algorithm 4 was applied to select actions from within the two clusters when creating action sets during exploration. The resulting comparative graphs in Figure 3 over four randomly generated new tasks show an improvement in learning times, similar to the action elimination.

The significant improvement in convergence times after structure is applied, even with the randomness of choosing a cluster and then selecting actions within the cluster, highlights just how much time is typically lost by the agent in an $\epsilon-$greedy exploration process.

The performance of the clustering approach could be improved by knowing which cluster to select based on the state, for example selecting from the door-related actions when at a door state and navigation-related actions otherwise. This contextual approach should have a positive impact on learning, however it is interesting to observe that just an analysis of the action space can produce a useful reduction in training time that is once again invariant across tasks.

7 Conclusion and Future Work

This paper proposed two approaches for using structure to improve sample efficiency during exploration in concurrent action environments including action elimination and clustering. Count data from successful trajectories were used as a basis for extracting prior knowledge across multiple tasks in the same domain. The priors were used to enhance a Q-learning algorithm and showed significant improvements in policy convergence times. Given that these are context free mechanisms there is room for improvement by bringing in context.

Some of the key limitations to these approaches include the need for a pre-training phase which means the methods are not online. There is also a need for successful trajectory data which limits these approaches to environments that are solvable by other means. Other work such as Tennenholtz et al. [15] and Harmer et al. [5] use demonstrator or expert trajectories which could work here too.

Future work would look into online methods that would remove the pre-training phase and not require solved trajectories. This will impact the scalability of these approaches. Secondly the use of context to better select clusters of actions will be considered.

References

1. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 4148–4152. IJCAI'15, AAAI Press (2015), http://dl.acm.org/citation.cfm?id=2832747.2832830
2. Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., Thomas, P.: Learning action representations for reinforcement learning. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97,

pp. 941–950. PMLR, Long Beach, California, USA (09–15 Jun 2019), http://proceedings.mlr.press/v97/chandak19a.html
3. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. AAAI/IAAI (1998)
4. Dimakopoulou, M., Van Roy, B.: Coordinated exploration in concurrent reinforcement learning. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1271–1279. PMLR, Stockholmsmässan, Stockholm Sweden (2018)
5. Harmer, J., Gisslén, L., del Val, J., Holst, H., Bergdahl, J., Olsson, T., Sjöö, K., Nordin, M.: Imitation learning with concurrent actions in 3d games. In: 2018 IEEE Conference on Computational Intelligence and Games, CIG 2018, Maastricht, The Netherlands, August 14-17, 2018. pp. 1–8 (2018). https://doi.org/10.1109/CIG.2018.8490398, https://doi.org/10.1109/CIG.2018.8490398
6. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing **17**(4), 395–416 (Dec 2007). https://doi.org/10.1007/s11222-007-9033-z, http://dx.doi.org/10.1007/s11222-007-9033-z
7. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013), http://dblp.uni-trier.de/db/journals/corr/corr1301.html/abs-1301-3781
8. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. pp. 1928–1937. ICML'16, JMLR.org (2016), http://dl.acm.org/citation.cfm?id=3045390.3045594
9. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems 14, pp. 849–856. MIT Press (2002), http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm.pdf
10. Rohanimanesh, K.: Concurrent decision making in Markov decision processes. Citeseer (2006)
11. Rosman, B., Ramamoorthy, S.: Action priors for learning domain invariances. IEEE Trans. Auton. Ment. Dev. **7**(2), 107–118 (Jun 2015)
12. Sharma, S., Suresh, A., Ramesh, R., Ravindran, B.: Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. CoRR **abs/1705.07269** (2017), http://arxiv.org/abs/1705.07269
13. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Nature **550**, 354– (Oct 2017), http://dx.doi.org/10.1038/nature24270
14. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998), http://www.cs.ualberta.ca/ sutton/book/the-book.html
15. Tennenholtz, G., Mannor, S.: The natural language of actions. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 6196–6205. PMLR, Long Beach, California, USA (09–15 Jun 2019), http://proceedings.mlr.press/v97/tennenholtz19a.html

16. Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A.S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T.P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., Tsing, R.: Starcraft II: A new challenge for reinforcement learning. CoRR **abs/1708.04782** (2017), http://arxiv.org/abs/1708.04782
17. Wang, H., Yu, Y.: Exploring multi-action relationship in reinforcement learning. In: Pacific Rim International Conference on Artificial Intelligence. pp. 574–587. Springer (2016)
18. Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D.J., Mannor, S.: Learn what not to learn: Action elimination with deep reinforcement learning. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 3562–3573. Curran Associates, Inc. (2018), http://papers.nips.cc/paper/7615-learn-what-not-to-learn-action-elimination-with-deep-reinforcement-learning.pdf