CANDID DAC: Leveraging Coupled Action Dimensions with Importance Differences in DAC

Philipp Bordne[1] **M. Asif Hasan**[1] **Eddie Bergman**[1] **Noor Awad**[1] **André Biedenkapp**[1]

[1]University of Freiburg

Abstract High-dimensional action spaces remain a challenge for dynamic algorithm configuration (DAC). Interdependencies and varying importance between action dimensions are further known key characteristics of DAC problems. We argue that these Coupled Action Dimensions with Importance Differences (CANDID) represent aspects of the DAC problem that are not yet fully explored. To address this gap, we introduce a new white-box benchmark within the DACBench suite that simulates the properties of CANDID. Further, we propose sequential policies as an effective strategy for managing these properties. Such policies factorize the action space and mitigate exponential growth by learning a policy per action dimension. At the same time, these policies accommodate the interdependence of action dimensions by fostering implicit coordination. We show this in an experimental study of value-based policies on our new benchmark. This study demonstrates that sequential policies significantly outperform independent learning of factorized policies in CANDID action spaces. In addition, they overcome the scalability limitations associated with learning a single policy across all action dimensions. The code used for our experiments is available under `https://github.com/PhilippBordne/candidDAC`.

1 Introduction

In the Dynamic Algorithm Configuration (DAC) problem (Biedenkapp et al., 2020) hyperparameters must be adjusted on-the-fly. A significant portion of these are either categorical or discrete, posing a challenge for reinforcement learning (RL), the common approach to solving DAC (Adriaensen et al., 2022), as they result in a combinatorial explosion of the joint action space. The DAC problem is further complicated by interaction effects between hyperparameters (Hutter et al., 2014; van Rijn and Hutter, 2018; Usmani et al., 2023) with varying importance of hyperparameters (Hutter et al., 2014; Moosbauer et al., 2021; Mohan et al., 2023; Biedenkapp et al., 2018). In this paper, we will refer to these properties as Coupled ActioN-Dimensions with Importance Differences (CANDID). This work investigates how CANDID influences the performance of RL algorithms and lays the ground for the development of better methods to tackle the DAC problem. To do so, we introduce a new CANDID benchmark derived from the original Sigmoid benchmark (Biedenkapp et al., 2020). We believe that this captures the complexities associated with the high-dimensionality of action spaces in DAC and other control domains more comprehensively. We use this benchmark to evaluate RL algorithms that learn policies per action dimension in factored action spaces (Sharma et al., 2017; Xue et al., 2022). In particular, we implement two algorithm variants to learn sequential policies (Metz et al., 2017). We compare them against a single agent baseline as typically used in DAC as well as a multi-agent baseline. Our results suggest that under the CANDID properties, sequential policies can coordinate action selection between dimensions while avoiding combinatorial explosion of the action space. Our initial results encourage an extended study of sequential policies for DAC.

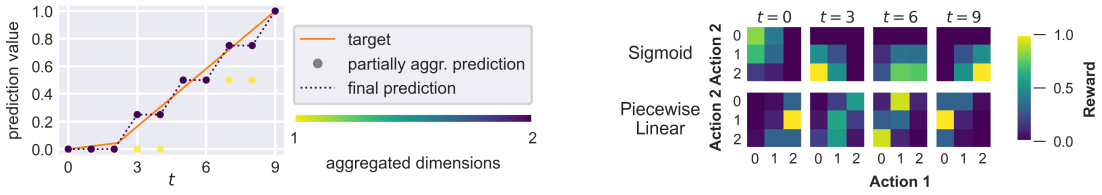
2 Related Work

Hyperparameter importances and interactions in the AutoML landscape. Hyperparameter importance is a key topic in AutoML. Tools like fANOVA (Hutter et al., 2014) quantify the importance of

individual hyperparameters and their interactions. These tools guide the tuning of hyperparameters and the examination of pipeline components in post-hoc analyses (van Rijn and Hutter, 2018). To our knowledge, the importance and interactions of hyperparameters have not yet been used as structural information to solve DAC problems more effectively.

Solving high-dimensional action spaces in RL. Large discrete or categorical action spaces pose a significant challenge for RL, as well as DAC by RL (Biedenkapp et al., 2022). Factored action space representations (FAR) address this challenge by learning policies per action dimension (Sharma et al., 2017). Based on Deep Q-Networks (DQN) (Mnih et al., 2015), Metz et al. (2017) introduced sequential policies that control one action dimension at a time and condition on previously selected actions. This approach has been extended to more RL-algorithms (Pierrot et al., 2021). We further develop this concept by ordering the sequential policies based on the importance of action dimensions and propose an adaptation of its training algorithm inspired by sequential games. Other research has approached high-dimensional action spaces as multi-agent learning problems, where each action dimension is controlled by a distinct agent. For example, Xue et al. (2022) trained individual agents per hyperparameter to dynamically tune a multi-objective optimization algorithm. In their study, well-established MARL algorithms such as VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) were utilized to coordinate learning in the multi-agent setting. As we aim to explicitly exploit the CANDID property of DAC action spaces through sequential action selection, we focus solely on the simplest MARL baseline which is independent learning of agents. Analogous to MARL research, allowing sequential policies to observe previously selected actions acts as a communication scheme between agents. This approach is tailored to the structure revealed by the importance of action dimensions and their interaction effects. For further insights into the state-of-the-art of MARL, we refer interested readers to the survey of Huh and Mohapatra (2023).

3 Piecewise Linear Benchmark



(a) (b)

Figure 1: (a) Example of a prediction task on a 2D Piecewise Linear instance; (b) Comparison of its reward surface against a 2D Sigmoid instance at 4 different time steps.

We implement the Piecewise Linear benchmark within DACBench (Eimer et al., 2021), building on the Sigmoid benchmark that models a DAC problem with a high-dimensional action space (Biedenkapp et al., 2020). In a M-D Sigmoid, the task is to predict the values of M individual Sigmoid curves over $T = 10$ time steps, with n^m_{act} action choices for dimension m. Its reward function is defined as $r_t = \prod_{m=1}^{M} 1 - \texttt{pred\_error}(a^m_t)$, which can be maximized by minimizing the prediction errors independently per action dimension. In contrast the Piecewise Linear benchmark aggregates all action dimensions through a weighted sum to predict on a *single* target function (Figure 1a):

$$\texttt{pred}(a^{1:M}_t) = \frac{a^1_t}{n^1_{\text{act}} - 1} + \sum_{m=2}^{M} w_m \left(\frac{a^m_t}{n^m_{\text{act}} - 1} - \frac{1}{2}\right) \tag{1}$$

This introduces coupling and importance differences between action dimensions, emulating the CANDID setting, and requires joint optimization of action dimensions (Figure 1b). Instead of

Sigmoid curves, we sample 2-segment piecewise linear functions as prediction targets to avoid constant target function values featured by many Sigmoid instances. We present more details such as reward signal definition and the train and test dataset in Appendix A.

4 Controlling CANDID Action Spaces with Sequential Policies

The aim of RL is to learn the optimal policy for a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$ (Sutton and Barto, 2018). After factorizing a M-dimensional action space $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_M$ into 1-dimensional action spaces $\mathcal{A}_1, ..., \mathcal{A}_M$ we can learn a policy per action dimension. We will refer to such approaches as factorized policies. Sequential policies build on the idea of extending the original MDP $\mathcal{M}$ to a sequential MDP (sMDP) by introducing substates $([s_t,], [s_t, a_t^1], ..., [s_t, a_t^{1:M-1}])$ to include an action selection *process* for all action dimensions at the current time step t (Metz et al., 2017). A comprehensive formalization of these MDP reformulations is given in Appendix B.1.

We choose sequential policies to solve CANDID action spaces for two reasons. (I) Sequential policies are able to condition subsequent actions on already selected actions and thus to learn about the coupling between action dimensions. (II) The different importances of action dimensions induce an order for the selection process: By selecting the most important action first, this information is available when controlling all other action dimensions at the current time step. We implemented two different algorithms to learn sequential policies, which differ in the TD-updates (see, Equation (2)). The first is a simplified version of SDQN (simSDQN) that omits the upper Q-network compared to the implementation of Metz et al. (2017). This approach can be interpreted as a hybrid of tabular and function-approximation Q-learning. We introduce an implicit state that denotes the current stage in the action selection process and add a tabular entry for each of the M stages. The second approach can be interpreted as multiple agents playing a sequential game (Sequential Agent Q-Learning = SAQL) and selecting a_t^m is the turn of the m-th agent, an approach inspired by learning equilibria in Stackelberg games (Gerstgrasser and Parkes, 2023). As our setting is fully cooperative, all agents receive the same true reward. We provide a more detailed explanation in Appendix B.3.

$$\text{target}^m_{\text{SAQL}} = r_t + \gamma \max_{a^m} Q^m([s_{t+1}, a_{t+1}^{1:m-1}], a^m) = r_t + \gamma V^m(s_{t+1}^m) \tag{2a}$$

$$\text{target}^m_{\text{simSDQN}} = \begin{cases} \max_{a^{m+1}} Q^{m+1}([s_t, a_t^{1:m}], a^{m+1}) = V^{m+1}(s_t^{m+1}), & 1 \leq m \leq M-1 \\ r_t + \gamma \max_{a^1} Q^1([s_{t+1},], a^1) = r_t + \gamma V^1(s_{t+1}^0), & m = M \end{cases} \tag{2b}$$

5 Experimental Setup

We compared sequential policies on the Sigmoid and Piecewise Linear benchmark against two baselines: (I) Double DQN (DDQN) (van Hasselt et al., 2016) as baseline of a single-agent policy controlling all action dimensions simultaneously; (II) Independent Q-Learning (IQL) (Tampuu et al., 2015) as multi-agent baseline controlling all action dimensions independently. Note that IQL can be seen as an ablation of SAQL without communication between individual agents. We provide a detailed comparison of the evaluated algorithms in Appendix B.3.

Additional action dimensions of the Piecewise Linear benchmark enable higher rewards through more accurate predictions. As static baseline, we calculated the reward achievable by predictions based solely on the first and most important action dimension (*optimal (1D)*). Outperforming this baseline indicates effective coordination of action dimensions.

We selected hyperparameters on a per-algorithm basis on the training instance set of the 5D Sigmoid benchmark. To this end, we evaluated each algorithm on a portfolio of 100 randomly sampled hyperparameter configurations. Each evaluation consisted of 10 random seeds that we aggregated through the median (Agarwal et al., 2021). We provide the configurations together with the model architectures in Appendices C & D and the used computing resources in Appendix E.

To induce importance differences, we define the weights per action dimension as $w_m = \lambda^{m-1}$ for $m \in \{2, ..., M\}$. We set the importance decay $\lambda = 0.5$ for all experiments and report the results

for further importance decays in Appendix F. To assess scaling with action space size, we evaluated the approaches on varying action space dimensionality (`dim`, or M) and actions per dimension (`n_act`). We provide details on the train and test dataset in Appendix A.

6 Results and Discussion

Are Sequential Policies Beneficial for CANDID DAC? To answer this question, we compare the best performing approaches on the 5D Sigmoid without CANDID properties and the 5D Piecewise Linear benchmark with CANDID properties. The performance of the multi-agent baselines (IQL) is notably poor in the CANDID setting and performs best in other scenarios, as illustrated in Figure 2. Even on the 2D Piecewise Linear benchmark, where all other evaluated algorithms achieve near-perfect solutions within a fraction of the total training episodes, IQL lags behind. This demonstrates the need for a mechanism of coordination between action dimensions with interaction effects and becomes especially evident when comparing performances of IQL and SAQL.

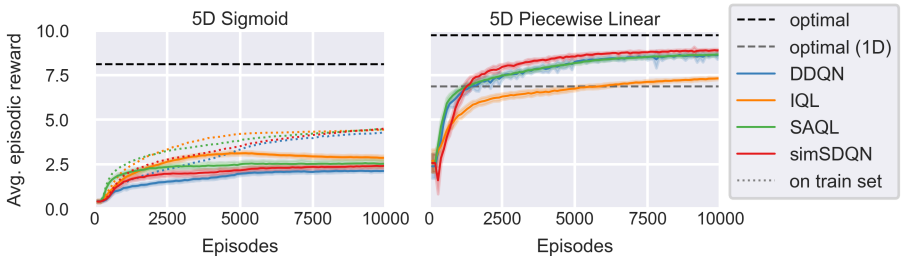


Figure 2: Average episodic rewards (mean, std from 20 seeds) on the test sets of 5D Sigmoid and Piecewise Linear benchmark ($\lambda = 0.5$ and `n_act` $= 3$). Generalization error on 5D Sigmoid results from hyperparameter selection on its training set.

Can Sequential Policies Scale to Larger Configuration Spaces? To answer this, we vary the number of discrete action choices per dimension (n_act) and the number of action dimensions. Although being slightly outperformed by simSDQN and matched by DDQN on the 5D Piecewise Linear benchmark, the performance of SAQL (as IQL) remains stable when increasing the dimension of the action space to 10 (Figure 3, upper row). simSDQN is notably impacted by the expansion of action space dimensionality. This outcome is likely attributed to the reward signal being solely observed in the TD-update of the last, i.e. the least important, action dimension. As a result, reward information must be propagated through more Q-networks before it reaches the most important policies, which is required for them to identify their optimal actions. DDQNs performance also decreases, as adding action dimensions leads to exponential growth of its action space. Increasing the number of discrete action choices per action dimension has a similar effect on DDQN as it still results in polynomial growth in size of its action space. Factorized policies are not negatively affected by this change, as can be seen in the bottom row of Figure 3, as it results only in a linear growth in size of their action spaces.

We further observed that the degree of importance differences between action dimensions has minimal impact on the relative performances of the evaluated algorithms, as detailed in Appendix F. In another ablation, we found that inverting the importance order of action selection in sequential policies slightly reduces their performance, which supports our initial intuition about the effectiveness of selecting the most important action first (Appendix G).

7 Limitations

This exploratory study is limited by its reliance on a white-box benchmark emulating the CANDID properties. It may not fully encapsulate the complexity of real-world applications, which could

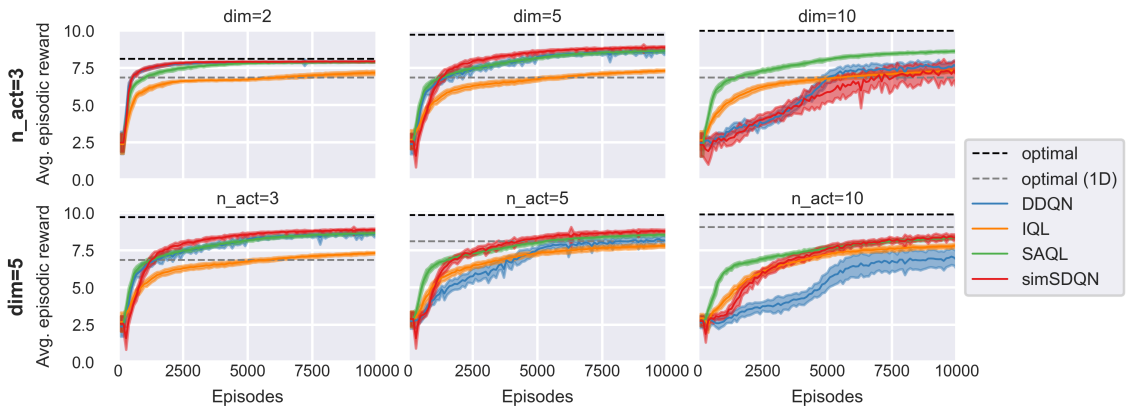


Figure 3: Experiments investigating scaling behavior of algorithms. First row keeps number of actions per action dimension fixed at `n_act` = 3 and varies dimensionality `dim` of action space. Second row keeps `dim` fixed and varies `n_act`. Both experiments keep importance decay $\lambda = 0.5$ fixed. Rewards from test set (mean, std from 20 seeds).

limit the general applicability of the findings. It is also limited to the most basic multi-agent baseline, IQL and comparison against more advanced algorithms such as QMIX would allow a better contextualization of sequential policies' performance within the SOTA. Several of the evaluated algorithms achieved near optimal performance. This might limit its suitability for the validation of improvements to these algorithms. A major drawback of our implementation of sequential policies as compared to traditional multi-agent algorithms is that the dimensionality of their observation space grows linearly with the number of interacting action dimensions and it is not straightforward to add policies later on.

8 Broader Impact Statement

While this work was specifically directed towards DAC, the challenge of controlling numerous interacting inputs is ubiquitous in most complex real-world systems. As such, the introduced approach could be applied in settings where communication between components of these systems is possible. Some examples are robots with multiple joints, smart buildings or grids. Application of our approach would require prior analysis of the structure of the control problem, in particular, to identify interacting control inputs and their relative importance. Sequential policies require, like other factorized approaches, the learning of multiple policies. This can lead to an increased demand for training and computing time and memory.

9 Conclusions

In this report, we introduced coupled action dimensions with importance differences (CANDID) as an under-researched challenge for reinforcement learning & DAC and provided the Piecewise Linear benchmark for assessing RL algorithms in these settings. In our experimental study, we have shown that sequential policies are a promising technique to address this challenge. In future work, we plan to apply SAQL to real-world inspired benchmarks and to compare it against more advanced MARL baselines that use value function factorization such as QMIX. As we see value function factorization to be mainly orthogonal to SAQL, we also plan to combine the two approaches. Moreover, we plan to improve agent coordination by exploring learned message passing, inspired by existing approaches like Huang et al. (2020). This strategy aims to prevent observation space growth with action dimensionality, improving scalability.

Acknowledgements. The authors acknowledge support by the state of Baden-Württemberg through bwHPC. Noor Awad and André Biedenkapp acknowledge funding from The Carl Zeiss Foundation through the research network "Responsive and Scalable Learning for Robots Assisting Humans" (ReScaLe) of the University of Freiburg. We acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under SFB 1597 (SmallData), grant number 499552394.

References

Adriaensen, S., Biedenkapp, A., Shala, G., Awad, N., Eimer, T., Lindauer, M., and Hutter, F. (2022). Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research (JAIR)*, 75:1633–1699.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. G. (2021). Deep reinforcement learning at the edge of the statistical precipice. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS'21)*, pages 29304–29320.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. (2020). Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In Giacomo, G. D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., and Lang, J., editors, *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI'20)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 427–434. IOS Press.

Biedenkapp, A., Dang, N., Krejca, M. S., Hutter, F., and Doerr, C. (2022). Theory-inspired parameter control benchmarks for dynamic algorithm configuration. In Fieldsend, J. E. and Wagner, M., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'22)*, pages 766–775. ACM.

Biedenkapp, A., Marben, J., Lindauer, M., and Hutter, F. (2018). CAVE: configuration assessment, visualization and evaluation. In Battiti, R., Brunato, M., Kotsireas, I. S., and Pardalos, P. M., editors, *Proceedings of the 12th International Conference on Learning and Intelligent Optimization (LION'18)*, volume 11353 of *Lecture Notes in Computer Science*, pages 115–130. Springer.

Eimer, T., Biedenkapp, A., Reimer, M., Adriaensen, S., Hutter, F., and Lindauer, M. (2021). Dacbench: A benchmark library for dynamic algorithm configuration. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 1668–1674. ijcai.org.

Gerstgrasser, M. and Parkes, D. C. (2023). Oracles & followers: Stackelberg equilibria in deep multi-agent reinforcement learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *Proceedings of the International Conference on Machine Learning ICML'23*, volume 202 of *Proceedings of Machine Learning Research*, pages 11213–11236. PMLR.

Huang, W., Mordatch, I., and Pathak, D. (2020). One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 119 of *Proceedings of Machine Learning Research*, pages 4455–4464. PMLR.

Huh, D. and Mohapatra, P. (2023). Multi-agent reinforcement learning: A comprehensive survey. *arXiv:2312.10256 [cs.MA]*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31th International Conference on Machine Learning (ICML'14)*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 754–762. JMLR.org.

Leyton-Brown, K. and Shoham, Y. (2008). *Essentials of Game Theory: A Concise Multidisciplinary Introduction*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. (2017). Discrete sequential prediction of continuous actions for deep RL. *arXiv:1705.05035 [cs.LG]*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mohan, A., Benjamins, C., Wienecke, K., Dockhorn, A., and Lindauer, M. (2023). AutoRL hyperparameter landscapes. In Faust, A., Garnett, R., White, C., Hutter, F., and Gardner, J. R., editors, *Proceedings of the International Conference on Automated Machine Learning (AutoML'23)*, volume 224 of *Proceedings of Machine Learning Research*, pages 13/1–27. PMLR.

Moosbauer, J., Herbinger, J., Casalicchio, G., Lindauer, M., and Bischl, B. (2021). Explaining hyperparameter optimization via partial dependence plots. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS'21)*, pages 2280–2291.

Pierrot, T., Macé, V., Sevestre, J.-B., Monier, L., Laterre, A., Perrin, N., Beguir, K., and Sigaud, O. (2021). Factored action spaces in deep reinforcement learning. OpenReview.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., and Whiteson, S. (2018). QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR.

Sharma, S., Suresh, A., Ramesh, R., and Ravindran, B. (2017). Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. *arXiv:1705.07269 [cs.LG]*.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In André, E., Koenig, S., Dastani, M., and Sukthankar, G., editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'18)*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2 edition.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. (2015). Multiagent cooperation and competition with deep reinforcement learning. *arXiv:1511.08779 [cs.AI]*.

Usmani, I. A., Qadri, M. T., Zia, R., Alrayes, F. S., Saidani, O., and Dashtipour, K. (2023). Interactive effect of learning rate and batch size to implement transfer learning for brain tumor classification. *Electronics*, 12(4).

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In Schuurmans, D. and Wellman, M., editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, page 2094–2100. AAAI Press.

van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In Guo, Y. and Farooq, F., editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*, pages 2367–2376. ACM.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Xue, K., Xu, J., Yuan, L., Li, M., Qian, C., Zhang, Z., and Yu, Y. (2022). Multi-agent dynamic algorithm configuration. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Proceedings of the 35th Annual Conference on Neural Information Processing Systems (NeurIPS'22)*.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The benchmark design reflects the stated challenges (section 3) and experimental results and discussion show superior performance of our approach (section 6)

   (b) Did you describe the limitations of your work? [Yes] see section 7

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] see section 8

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes] We did several proofreads and use colorblind-friendly colors in our plots.

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [No] We used the same benchmarks and instance sets for all methods. We trained all methods on the same type of CPU but for the DDQN experiments with (dim=10, n_act=3) and (dim=5, n_act=10) we had to use a GPU due to the size of the resulting Q-networks. However this shouldn't affect our comparisons since we compared progress over the number of episodes and also limited training by the number of episodes and not in training time.

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] We describe the hyperparameter search process qualitatively in section 5 and the used hyperparameters in appendix C. The benchmark and its instances are part of the published code, the search for hyperparameters is conducted through the main experimental script and comprehensible from there.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] We evaluated every (method,benchmark)-pair on 20 random seeds. During hyperparameter selection we evaluated every method on 10 random seeds per configuration.

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] We included std in our plots.

(e) Did you report the statistical significance of your results? [No] This study is exploratory and we believe our conclusions are justifiable by the plots including std.

(f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] This was not in our scope of finding suitable RL algorithms for DAC.

(g) Did you compare performance over time and describe how you selected the maximum duration? [No] We did no such comparisons.

(h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] see appendix E

(i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We did limited ablations: comparing IQL and SAQL (section 6) and by validating the order of sequential policies (appendix 9)

3. With respect to the code used to obtain your results...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] We provided a link to the repository.

(b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] Most of the experiments are small-scale and can be executed locally. We provide examples how to execute them locally.

(c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [Yes] We use hydra to define experimental setups and provide a description on how to that in the README. We also focused on documentation of the main script. The source code for our benchmark and algorithms is mostly typed but we are still working on improving it until publication.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] The metrics in the plots are included in a .csv in the directory analysis/run_data within the repository.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [Yes] In the directory analysis/ we provide the notebooks used to generate the plots.

4. If you used existing assets (e.g., code, data, models)...

(a) Did you cite the creators of used assets? [Yes] We used DACBench and cite it in our paper

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] No such material was used.

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No such data was used.

5. If you created/released new assets (e.g., code, data, models)...

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] license in repository

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] availabe on GitHub

6. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

7. If you included theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

A Details on Piecewise Linear Benchmark

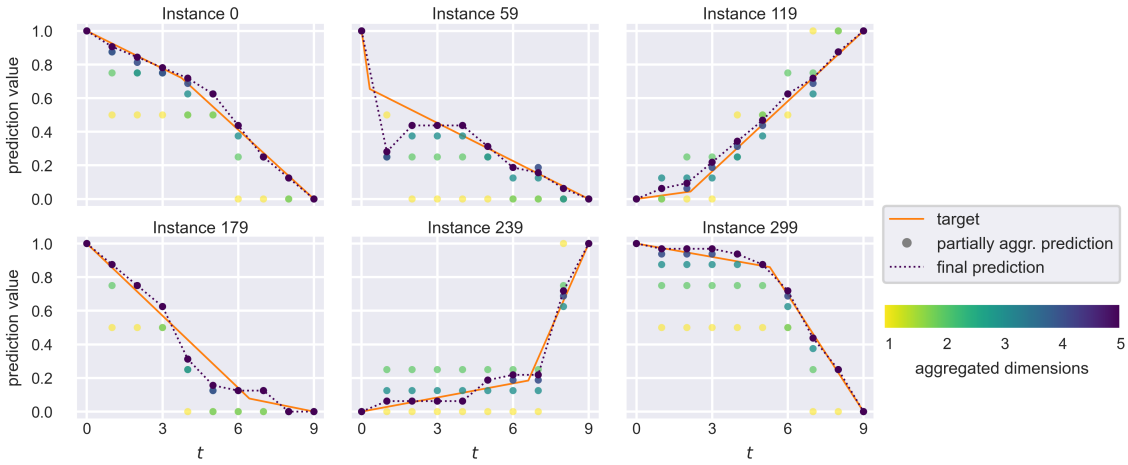


Figure 4: Several examples of test instances of the Piecewise Linear benchmark and predictions from a policy learned through SAQL.

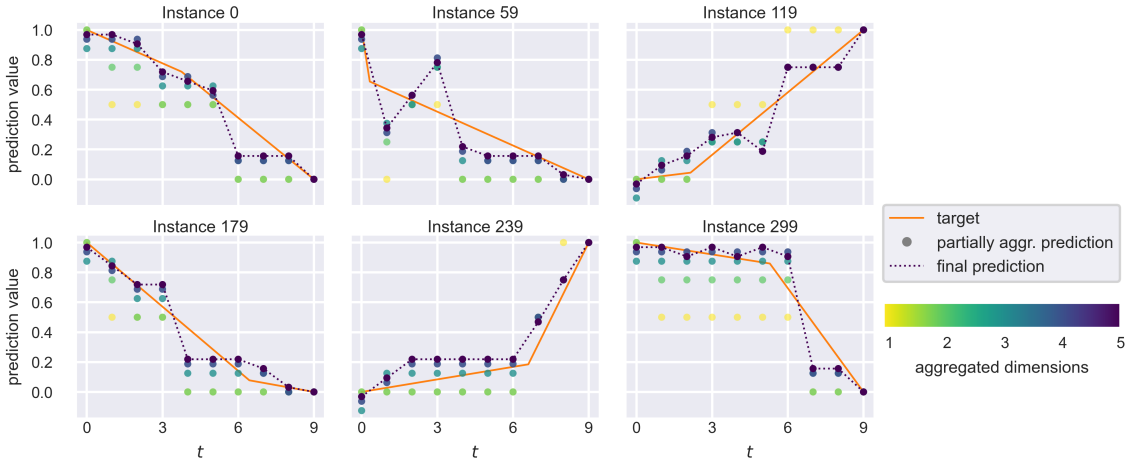


Figure 5: Several examples of test instances of the Piecewise Linear benchmark and predictions from a policy learned through IQL.

Train and Test Instance Datasets. The Piecewise Linear benchmark's training and test datasets $\mathcal{I}_{\text{train}}$ and $\mathcal{I}_{\text{test}}$ consist of 300 different target function instances each. Instances for both datasets were generated in a two-step process. First we randomly sample an intermediate point (x, y), where $x \in [0, 9]$ and $y \in [0, 1]$. Next, a random choice determines whether to establish a connection between $(0, 0)$ and $(9, 1)$ (increasing) or between $(0, 1)$ and $(9, 0)$ (decreasing) through the sampled intermediate point. Together this defines the 2 piecewise linear segments of our target function. When interacting with the Piecewise Linear benchmark, one episode consists of predicting the function values of a specific target function instance over $T = 10$ time steps. During training we reset the environment after finishing an episode and continue with predicting on the next target function instance (round-robin). We note that this defines a contextual MDP (cMDP) as it is proposed for the DAC framework (Biedenkapp et al., 2020). We also note that the described

instances are independent from the number of action dimensions and actions per action dimension. Hence we used the same train and test instance datasets for all our experiments.

Figure 4 visualizes different instances from the generated instance set, along with the resulting predictions obtained by aggregating an increasing number of action dimensions through a sequential policy. It shows that as more action dimensions are aggregated, the prediction accuracy generally improves. Figure 6 illustrates, for a single time step, how a sequential policy progressively approximates the value of the prediction target more accurately as it aggregates more dimensions. In contrast, Figure 5 shows how independent policies often fail to coordinate.

Observation Space. Similar to the Sigmoid benchmark (Biedenkapp et al., 2020) the observation space consists of information about the current instance $i \in \mathcal{I}$, the remaining number of steps for the current instance $T - t$ and the actions selected at the previous time step. The current instance is uniquely defined by the coordinates (x, y) of the intermediate point and a bit b which determines whether the target function is increasing or decreasing. Thus, the observation vector of policies in a single policy setting (DDQN) and in a factorized but non-sequential policy setting (IQL) is defined as $o_t = [T - t, x_i, y_i, b_i, a^1_{t-1}, ..., a^M_{t-1}]$. The observation of an atomic policy m in the sequential policy setting (SAQL, simSDQN) additionally includes actions selected by its predecessors at the current time step: $o^m_t = [T - t, x_i, y_i, b_i, a^1_{t-1}, ..., a^M_{t-1}, a^1_t, ..., a^{m-1}_t]$.

Reward Computation. To compute the reward at step t, we first calculate the prediction error as the absolute difference between the aggregated prediction $\texttt{pred}(a^{1:M}_t)$ and the value of the piecewise linear function $\texttt{pl}(t)$ at the current point in time:

$$\texttt{pred\_error}(a^{1:M}_t) = |\texttt{pred}(a^{1:M}_t) - \texttt{pl}(t)| \tag{3}$$

The aggregated prediction $\texttt{pred}(a^{1:M}_t)$ is defined in Equation (1). To incentivize learning across all action dimensions, including those with less significant contributions, we define an exponentially decaying reward signal $r_t = e^{-c \cdot \texttt{pred\_error}(a^{1:M}_t)}$. This formulation ensures that selecting the first action optimally is necessary to obtain high rewards, but it might not be sufficient, as the exponentially decaying reward puts more emphasis on a *precise* prediction. For our experiments, we have chosen $c = 4.6$.

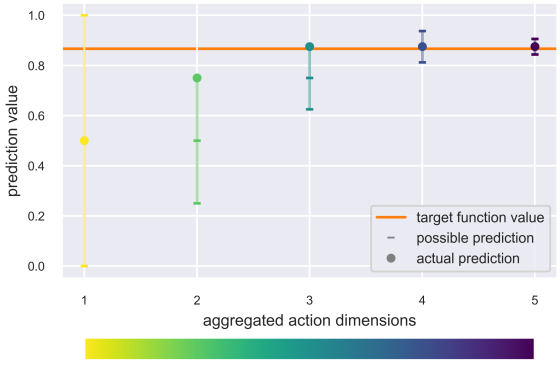


Figure 6: Illustration of action selection by a sequential policy trained through SAQL on the Piecewise Linear benchmark at time step $t = 5$ of benchmark instance 299. It demonstrates how the prediction gets iteratively fine-tuned to fit the target value.

B MDP Reformulations and Algorithms in Detail

B.1 Action Space Factorization and MDP Reformulations

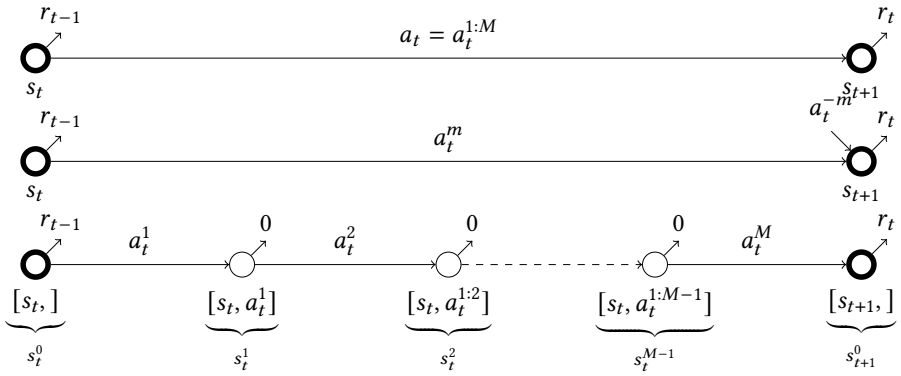


Figure 7: Transitions in the original MDP (upper), reformulated as a parallel MDP (pMDP, middle) and a sequential MDP (sMDP, below, adapted from Metz et al. (2017)), where $[\cdot, \cdot]$ represents augmented state vectors.

In this work we aim at solving Markov Decision Processes (MDP) with discrete action spaces, which can be defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$ where $\mathcal{S} \subseteq \mathbb{R}^N$ is the state space. $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_M$ is a factorization of $\mathcal{A} \subset \mathbb{N}^M$.

Parallel MDP. Given the factorization of $\mathcal{A}$, we define a parallel MDP (pMDP) $\mathcal{M}$ as an instance of a stochastic or Markov game $\mathcal{G} = (\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{A}$ is factorized per definition. Each $\mathcal{A}_m$ represents the actions available to one of the M players. R defines a pay-off function per agent (e.g. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^M$) (Leyton-Brown and Shoham, 2008) but we assume all agents to receive a shared reward to solve the original MDP $\mathcal{M}$. The distributions over the next state S_{t+1} and reward R_t not only depend on its own action but also the actions selected by all other players, denoted as a_t^{-m}. However in the pMDP these actions go unobserved.

Sequential MDP. Alternatively, given the factorization we define a sequential MDP (sMDP). We introduce intermediate steps into $\mathcal{M}$ which correspond to selecting one action after the other: $\mathcal{M}_{\text{seq}} = (\mathcal{S}_{\text{seq}}, \mathcal{A}_{\text{seq}}, P_{\text{seq}}, R_{\text{seq}})$, where $\mathcal{S}_{\text{seq}} = \langle \mathcal{S}_0, \mathcal{S}_1, ...\mathcal{S}_{M-1} \rangle$ and $\mathcal{A}_{\text{seq}} = \langle \mathcal{A}_1, ..., \mathcal{A}_M \rangle$ are ordered sets with $\mathcal{S}_0 = \mathcal{S}$ and $\mathcal{S}_m = \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_m$. Transitions happen periodically $\mathcal{S}_m \to \mathcal{S}_{(m+1) \bmod M}$, with reward given and time t incremented upon transitions into $s^0 \in \mathcal{S}_0$. In state $s^m \in \mathcal{S}^m$ action a^{m+1} can only be selected from $\mathcal{A}_{m+1}$ (Metz et al., 2017).

B.2 Background on (D)DQN

Before we translate the MDP reformulations into our modifications to DDQN (van Hasselt et al., 2016) we want to provide a brief introduction to Q-Learning and Deep Q-Networks and recommend the excellent text-book by Sutton and Barto (2018) for a more thorough introduction.

Q-Learning and TD-Updates. Q-Learning (Watkins and Dayan, 1992) is a widely used approach to learn optimal policies in MDPs by learning the state-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, (s, a) \mapsto Q(s, a)$. $Q(s, a)$ is an estimate of the accumulated (and potentially discounted) reward obtainable over an entire episode if choosing action a in state s. Given Q we define our policy $\pi(s) = \arg\max_a Q(s, a)$ and analogously we can compute the value function $V : \mathcal{S} \to \mathbb{R}, s \mapsto V(s)$ of a state as $V(s) = \max_a Q(s, a)$. The idea of Q-Learning is to update our estimate Q using temporal-difference (TD) updates through the Bellman Optimality Equation: $Q(s_t, a_t) = R_t + \gamma \max_a Q(S_{t+1}, a)$. R_t and S_{t+1} are random variables for the reward and next state we will end up with, if taking action a_t in state

s_t. Given an observed transition (s, a, r, s') in our MDP the TD-update of Q is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] = Q(s,a) + \alpha[r + \gamma V(s') - Q(s,a)] \quad (4)$$

Hence we define $r + \gamma \max_{a'} Q(s', a') = r + \gamma V(s')$ as our TD-target.

Q-Learning Through Function Approximation Using (D)DQN. In discrete state and action spaces of very limited size we might learn tabular entries per pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. For very big or infinite state and/or action spaces we have to resort to function approximation, for example through Deep Q-Networks (DQN, Mnih et al. (2015)). Here we update the parameters θ of our Q-function by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \quad (5)$$

Parameters θ^- represent target networks that are updated with a delay. D is a replay buffer where we store transitions (experiences) we collect while interacting with the environment. To update our Q-network we use mini-batches sampled from D. Analogous to Q-Learning the TD-target for DQN is $r + \gamma \max_{a'} Q(s', a'; \theta^-)$. We also note that in DQN Q-networks are mappings $q_\theta : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$. That is given a state s they assign a value to each action a or action combination in action vector $\mathbf{a} = a^{1:M} \in \mathbb{N}^M$ in case of M-dimensional action spaces. This means $Q(s, a; \theta) = q_\theta(s)[a]$.
Instead of the original DQN we implemented our approaches using Double DQN (DDQN, van Hasselt et al. (2016)) which is a minor extension to DQN but does not affect the presented conceptualization.

B.3 Learning Q-Networks in the MDP Reformulations

Algorithm	Underlying MDP	Q-network(s) to learn
DDQN	original MDP	$q : \mathcal{S} \rightarrow \mathbb{R}^{\lvert\mathcal{A}\rvert}$
IQL	parallel MDP	$\{q^m : \mathcal{S} \rightarrow \mathbb{R}^{\lvert\mathcal{A}_m\rvert} \mid m = 1, ..., M\}$
SAQL/simSDQN	sequential MDP	$\{q^m : \mathcal{S} \times_{i=1}^{m-1} \mathcal{A}_i \rightarrow \mathbb{R}^{\lvert\mathcal{A}_m\rvert} \mid m = 1, ..., M\}$

Table 1: Q-networks to learn in our evaluated algorithms.

In this section, we discuss in detail how to learn policies for different reformulations of the original MDP (Figure 7). For ease of presentation, we drop references to θ. Solving the original MDP is straightforward and requires the learning of a single policy. The related Q-network can be updated against the common TD-target.

For the parallel MDP, we apply Independent Q-Learning (IQL) (Tampuu et al., 2015), learning M policies, one per action dimension. Each associated Q-network takes the original state s_t as input and maps it to the Q-values of its respective action dimension. Since the actions of other agents go unobserved, they can be treated as unobserved (and nonstationary) environment dynamics. Consequently, we can update each of the M Q-networks independently using the shared, common reward, and the usual TD-target.

In the sequential MDP (Metz et al., 2017), policies can observe not only the current state s_t, but also the action dimensions already selected for that state. Accordingly, the Q-networks learn to map these augmented observations to the action values of their respective dimensions. We implemented two approaches to update these Q-networks, with different underlying interpretations. The first approach, Sequential Agent Q-Learning (SAQL), views the sequential MDP as a sequential stochastic game. In substate $s_t^{m-1} = [s_t, a^{1:m-1}]$, it is agent m's turn to choose an action, observing the actions already taken by other players in round t. Agent m will make its next observation in

round $t+1$, with some fellow players having already acted. Here, we apply the standard DDQN algorithm, limited to the respective action dimension, using an augmented observation space and the shared reward r_t, since the game setting is fully cooperative. IQL can be seen as an ablation of SAQL, with identical Q-networks and TD-targets, except for excluding the observation of other agents' actions.

The second approach, Simplified Sequential DQN (simSDQN), is mainly identical to the original proposal by Metz et al. (2017). Our modification is to omit the upper Q-network, because we couldn't successfully train under this setup. We refer the reader to the paper by Metz et al. (2017) for the role of the upper Q-network. In simSDQN, we explicitly solve the sequential MDP by updating our Q-Functions against the state-value of the next substate in the sMDP. Using M Q-networks can be viewed as tabular entries for each of the M substates recurring periodically. Table 1 lists the Q-networks to be learned for our different approaches, and Equation (6) formalizes the targets for the Q-network updates.

$$\text{target}_{\text{DDQN}} = r_t + \gamma \max_{\mathbf{a}} Q(s_{t+1}, \mathbf{a}) = r_t + \gamma V(s_{t+1}) \tag{6a}$$

$$\text{target}^m_{\text{IQL}} = r_t + \gamma \max_{a^m} Q^m(s_{t+1}, a^m) = r_t + \gamma V^m(s_{t+1}) \tag{6b}$$

$$\text{target}^m_{\text{SAQL}} = r_t + \gamma \max_{a^m} Q^m([s_{t+1}, a^{1:m-1}_{t+1}], a^m) = r_t + \gamma V^m(s^m_{t+1}) \tag{6c}$$

$$\text{target}^m_{\text{simSDQN}} = \begin{cases} \max_{a^{m+1}} Q^{m+1}([s_t, a^{1:m}_t], a^{m+1}) = V^{m+1}(s^{m+1}_t), & 1 \le m \le M-1 \\ r_t + \gamma \max_{a^1} Q^1([s_{t+1},], a^1) = r_t + \gamma V^1(s^0_{t+1}), & m = M \end{cases} \tag{6d}$$

C Hyperparameter Settings

Hyperparameters	**DDQN**	**IQL**	**SAQL**	**simSDQN**
Learning Rate (α)	1.0076e-4	3.2680e-5	7.7590e-5	3.0855e-4
Discount Factor (γ)	0.9349	0.9147	0.9086	0.9696
Start Exploration Rate (ϵ_{start})	0.2382	0.9289	0.4607	0.1341
Target Update Frequency	15	39	12	33
Target Soft Update Factor (τ)	0.2613	0.1258	0.6196	0.4765
Batch Size	220	63	67	105
End Exploration Rate (ϵ_{end})	0.01			
Exploration fraction (linear ϵ - decay)	0.5			
Optimizer	Adam[1]			
Replay Buffer Size	2500			

Table 2: Hyperparameters per method, used for our experiments.

D Policy architecture

We represented the Q-functions of our learned policies as 3 layer MLPs with ReLU activation functions. Note that for the factorized policies IQL, SAQL, simSDQN the number of policies to learn corresponds to the dimension of the benchmark. For all policies, we used shared numbers of hidden units: 120 and 84 in the first and second hidden layers, respectively. For DDQN and IQL the number of input units corresponds to the dimensionality of the observation space of the benchmark environment. For sequential policies SAQL and simSDQN the size of the input layer of Q^m is `dim_observation_space` + m for $m \in \{0, ..., M-1\}$ (see Appendix A). The output size for all factorized policies is the number of actions per action dimension `n_act` $\in \{3, 5, 10\}$, for DDQN it is the number of all possible action combinations over all action dimensions $\texttt{n\_act}^M = \texttt{n\_act}^{\texttt{dim}}$.

E Compute Resources

Unless otherwise stated, the experiments were run on nodes using a single CPU "Intel Xeon Gold 6230". For DDQN experiments with higher-dimensional action spaces (`dim` = 10) or more discrete action choices per action dimension (`n_act` = 10), the resulting Q-networks were substantially larger, necessitating the use of GPU accelerators. These experiments were executed on nodes equipped with "NVIDIA Tesla V100" GPUs.

F Different Importance Decays

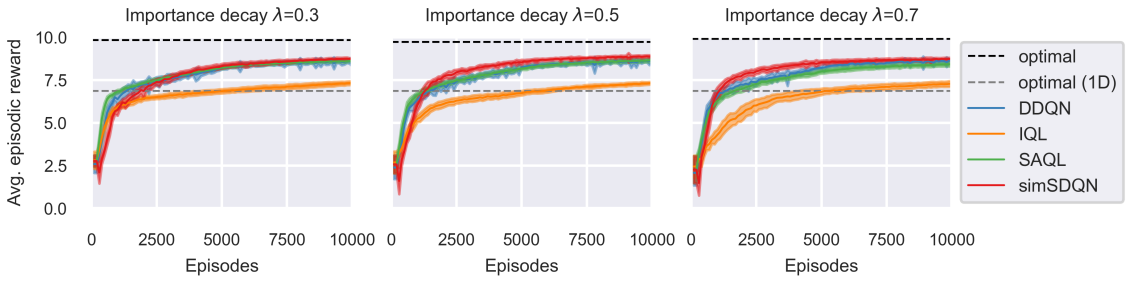


Figure 8: Average episodic test rewards (mean, std from 20 seeds) on the 5D Piecewise Linear benchmark, for different importance decays λ. Lower λ means importance is decreased more strongly from action dimension to action dimension.

G Reversed Importances

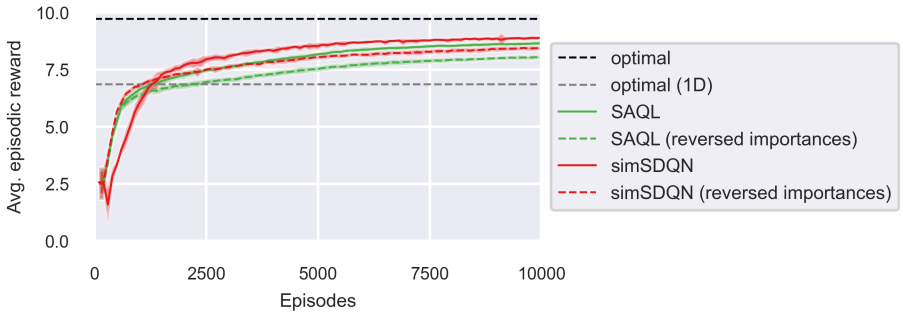


Figure 9: Average episodic test rewards (mean, std from 20 seeds) obtained by SAQL and simSDQN when selecting actions with descending or ascending (reversed) importance on 5D Piecewise Linear benchmark.

The ablation presented in Figure 9 confirmes our intuition regarding selecting actions in descending order of their importance for sequential policies and emphasizes the signficance of getting the order of importances right. This analysis also sheds light on a potential issue of simSDQN when facing higher dimensonal action spaces: In our design, rewards are only assigned when performing the TD-update for the Q-network responsible for the least important action dimension. This requires the propagation of reward information to more important action dimensions (for a conceptual illustration, refer to Figure 7). By reversing the order of action dimensions, the Q-network corresponding to the most important action dimension can be directly updated towards the reward signal. This leads to a noticeable speed up of learning in the initial training phase.