Learning Hierarchical Planning-Based Policies from Offline Data

Wöhlke, J.; Schmitt, F.; van Hoof, H.

DOI
10.1007/978-3-031-43421-1_29
Publication date
2023
Document Version
Final published version
Published in
Machine Learning and Knowledge Discovery in Databases: Research Track
License
Article 25fa Dutch Copyright Act (https://www.openaccess.nl/en/in-the-netherlands/you-share-we-take-care)
Link to publication

Citation for published version (APA):
Wöhlke, J., Schmitt, F., & van Hoof, H. (2023). Learning Hierarchical Planning-Based Policies from Offline Data. In D. Koutra, C. Plant, M. Gomes Rodriguez, E. Baralis, & F. Bonchi (Eds.), *Machine Learning and Knowledge Discovery in Databases: Research Track : European Conference, ECML PKDD 2023, Turin, Italy, September 18–22, 2023 : proceedings* (Vol. IV, pp. 489–505). (Lecture Notes in Computer Science; Vol. 14172), (Lecture Notes in Artificial Intelligence). Springer. https://doi.org/10.1007/978-3-031-43421-1_29

Learning Hierarchical Planning-Based Policies from Offline Data

Jan Wöhlke[1,2(✉)], Felix Schmitt[3], and Herke van Hoof[4]

[1] Bosch Center for Artificial Intelligence, 71272 Renningen, Germany
JanGuenter.Woehlke@de.bosch.com
[2] UvA-Bosch DELTA Lab, University of Amsterdam, Amsterdam, The Netherlands
[3] Robert Bosch GmbH, 70469 Stuttgart, Germany
Felix.Schmitt@de.bosch.com
[4] AmLab, University of Amsterdam, Amsterdam, The Netherlands
h.c.vanhoof@uva.nl

Abstract. Hierarchical policy architectures incorporating some planning component into the top-level have shown superior performance and generalization in agent navigation tasks. Cost or safety reasons may, however, prevent training in an online (RL) fashion with continuous environment interaction. We therefore propose HORIBLe-VRN, an algorithm to learn a hierarchical policy with a top-level planning-based module from pre-collected data. A key challenge is to deal with the unknown, latent high-level (HL) actions. Our algorithm features an EM-style hierarchical imitation learning stage, incorporating HL action inference, and a subsequent offline RL refinement stage for the top-level policy. We empirically evaluate HORIBLe-VRN in a long horizon, sparse reward agent navigation task, investigating performance, generalization capabilities, and robustness with respect to sub-optimal demonstration data.

Keywords: Learning from Demonstrations · Imitation Learning · Reinforcement Learning · Hierarchical RL

1 Introduction

Reinforcement learning (RL) is a popular paradigm to learn control policies from interaction with the environment or a good simulation thereof. So far, much RL research went towards this online learning setting, which led to remarkable success in mastering video games from visual inputs [27] or solving sparse reward robotic manipulation tasks [1]. However, especially in industrial settings, continuous online training may not be possible for availability (of test benches) or safety reasons. Furthermore, sufficiently accurate simulations may not be readily available or too expensive to create for very specific use cases.

A solution to this dilemma might be to collect a limited dataset of demonstrations and use it for offline policy learning. For this objective, the literature presents a variety of solutions ranging from basic behavioral cloning (BC) [3] imitation learning (IL) to offline RL algorithms [13,14,23]. Despite their differences, these methods are commonly designed to learn a "flat" policy from data.

For long horizon robot/vehicle navigation problems, Wöhlke et al. [38] showed that a hierarchical policy architecture with a top-level Value Refinement Network (VRN) policy that learns to refine a coarse prior plan shows superior performance. The planning component especially allows for better generalization towards new layouts, unseen during training. Fox et al. present in [10] a hierarchical imitation learning scheme, based on the options framework [32].

In this work, we combine the advantages of offline learning from data, not requiring environment interaction during training, with aforementioned effective hierarchical planning-based policy containing a VRN. Hence, we newly present the Hierarchical Offline Reinforcement and Imitation Batch Learning of VRN policies (HORIBLe-VRN) algorithm. It addresses the key challenge of latent high-level (HL) actions, which are the interface between policy levels but not part of the data (potentially collected with a flat policy), by performing hierarchical IL including HL action inference tailored to our architecture with a VRN. Furthermore, we add a novel offline RL refinement stage for the top-level VRN. In our empirical evaluation we investigate the following research hypotheses:

– *H.1*: Using a *planning-based* policy containing a VRN improves generalization (to unseen layouts) in the offline learning setting (as well).
– *H.2*: Learning a *hierarchical* policy architecture with temporal abstraction from offline data can further improve performance.
– *H.3*: Adding our offline RL *refinement stage* for the HL policy improves performance/robustness when the data is collected by sub-optimal policies.

2 Related Work

Our work combines aspects of imitation learning (IL), hierarchical (HRL), and offline RL. All of these are broad research fields in themselves. Hence, we refer the interested reader to recent surveys on IL [40], HRL [19], and offline RL [29].

We are particularly interested in hierarchical policy architectures that combine planning with RL. For robotic (navigation) tasks, these are commonly designed in a manager-worker "Feudal"-style [7,36], where a top-level planning policy sets sub-goals for some low-level, sub-goal-conditioned control policy: For the top-level planning, a sampling-based PRM planner [12], a differentiable ((M)VProp [28]) planning module [5], or value iteration (VI) with a learned transition model [37] have been used. In [38] VI is performed in a simpler state space abstraction, yielding a prior plan that is locally refined by a Value Refinement Network (VRN) using recent state information. The PAHRL approach [16] performs sub-goal planning on a graph of replay buffer states (like SoRB [9]). For the distance estimation, an (HAC-style [25]) Q-function hierarchy is trained via distributional RL, which is then also used for navigating the sub-goals.

Another related body of work focuses on learning hierarchical policy architectures from demonstrations. In [24] the algorithmic framework of "hierarchical guidance" is proposed, which allows utilizing different qualities of expert feedback to learn hierarchies of different combinations of IL and RL. Notably, hierarchical behavioral cloning (h-BC) is proposed. In contrast to our setting, where a static, pre-collected dataset is assumed, expert/environment interaction is possible during the policy learning. For h-BC, hierarchical demonstrations are assumed, which are generally not available. The HPIL approach [26] requires environment interaction as well. It performs an object-centric segmentation of demonstrations to then learn sub-policies for the segmented sub-tasks, in parallel, while simultaneously learning the meta-policy, leveraging the demonstrations for modified DDPG from Demonstrations [35]. For imitation learning parametrized hierarchical procedures (PHP), program-like structures that can invoke sub-procedures, variational inference methods are used in [11] to approximate a posterior distribution over the latent sequence of calls and terminations.

A line of research particularly relevant for our work investigates the hierarchical IL setting within the options framework [2,32], assuming the options to be latent variables. Modeling the data generation by an HMM, these approaches employ some EM-style [8] option inference. An early work [6], focusing on linear feature policies, employs a variant of the Baum-Welch (BW) algorithm [4] for the E-step to then perform a complete optimization in the M-step. Convergence guarantees for such an approach are investigated in [39]. An online BW algorithm, to process incoming data on the fly, is proposed in [15]. To allow for deep neural network policies, the DDO method in [10] uses an expectation gradient algorithm, similar to the ECG method in [30], to perform a gradient step on the policy parameters in the M-step. DDCO [22] extends DDO to continuous control settings and relaxes the pre-specification on the number of options. Option-GAIL [20] employs for the M-step option occupancy measurement matching, making use of adversarial learning to estimate the discrepancy between (inferred) expert and agent. A related online HRL approach is IOPG [31]: A policy gradient algorithm for the options framework assuming the options as latent variables and therefore containing a differentiable option inference procedure.

Offline learning of hierarchical planning-based policies has not been investigated, yet.

3 Technical Background and Problem Statement

3.1 Offline Learning Setting

Sequential decision-making problems can be modeled as Markov Decision Processes (MDPs). In this work, we look at goal-based MDPs, which we can denote as the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mathcal{S}_0, \mathcal{S}_g, \gamma, T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $\mathcal{P}(s'|s,a)$ are the dynamics of transitioning from a state s to a next state s' as a result of taking action a. Start states s_0 and goal states g are sampled from start and goal distributions $\mathcal{S}_0 \subseteq \mathcal{S}$ and $\mathcal{S}_g \subseteq \mathcal{S}$. The goal-dependent reward function $r(s,g)$ provides feedback and γ is the discount factor. The time horizon T determines the maximum number of steps to reach the goal.

In order to solve an MDP, we need to find a policy $\pi(a|s,g)$ that selects suitable actions a given the current state s and the goal state of the MDP g. Since we are in an offline learning setting, we cannot interact with the environment, hence have no access to $\mathcal{P}$ during training time. Instead, we need to learn π

from a dataset of collected demonstrations $\mathcal{D}$. We assume $\mathcal{D}$ to be a sequence of transition tuples from different demonstration episodes e. The tuples consist of state, action, next state, reward, termination flag, and goal state. With $|\mathcal{D}|$ being the size of the dataset (number of transition tuples), we can denote $\mathcal{D}$ as

$$\begin{aligned}\mathcal{D} = \big((s_0, a_0, s_1, r_0, d_0, g_0) \ldots \big(s_{|\mathcal{D}|-1}, a_{|\mathcal{D}|-1}, s_{|\mathcal{D}|}, r_{|\mathcal{D}|-1}, d_{|\mathcal{D}|-1}, g_{|\mathcal{D}|-1}\big) \big) \\ = ((s_t, a_t, s_{t+1}, r_t, d_t, g_t))_{t=0}^{|\mathcal{D}|-1} . \quad (1)\end{aligned}$$

3.2 Hierarchical Policy Architecture

We make use of a hierarchical policy architecture which is similar to the one presented in [37]. Figure 1 depicts the two-level hierarchy. It consists of a goal-conditioned high-level (HL) policy $\omega(o|z,g)$ that operates in a (discrete) state space abstraction with finite HL state space $\mathcal{Z}$ and finite HL action space $\mathcal{O}$. Transformations $f_{\mathcal{Z}}(s)$ and $f_{\mathcal{S}}(z)$, that are assumed known, transform low-level (LL) states s into HL states z and vice versa.

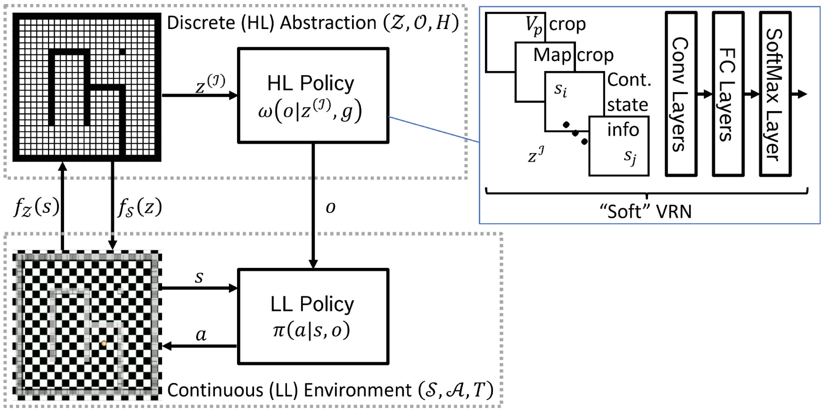


Fig. 1. Hierarchical policy architecture with "Soft" VRN.

The sub-goal conditioned low-level policy $\pi(a|s,o)$ directly interacts with the original MDP with state and action spaces $\mathcal{S}$ and $\mathcal{A}$. Please note that the LL policy receives a local sub-goal (tile) z_{sg} derived from the current HL state z and the chosen HL action o (reflecting a relative direction in navigation so that z_{sg} is the neighboring HL state z in that direction). So, we have $\pi(a|s,z_{sg})$. Since z_{sg} is a function of s and o, we denote the LL policy, for simplicity, as $\pi(a|s,o)$.

Furthermore, the HL policy operates on a different timescale than the LL policy. A new HL action o is selected when the HL state z changes or the sub-goal horizon H runs out. As a result, the LL policy experiences sub-episodes of maximum length H until the horizon T runs out or the MDP goal g is achieved.

We denote the parameters of the HL and LL policies as ϕ and θ, respectively.

3.3 Value Refinement Network

For improved generalization across different environments, we employ a specific Value Refinement Network (VRN) [38] for the HL policy ω. In short, the VRN is a specific convolutional neural network (CNN) architecture that locally refines a coarse prior plan $\widetilde{V}_p^{\mathcal{Z}}$. See top right of Fig. 1 for a depiction. The prior plan is obtained through value iteration (VI) in the HL abstraction, assuming HL transitions to be always successful, which results in a shortest path plan.

The VRN locally refines these prior values. Therefore, it receives a specific input representation $z^{\mathcal{I}} = f_{\mathcal{I}}\left(s, (z = f_{\mathcal{Z}}\left(s\right)), g, \Phi\right)$ based on the current LL (and HL) state, the goal, and the environment layout Φ so that we have $\omega^{\text{VRN}}\left(o|z^{\mathcal{I}}\right)$. In the depicted robot navigation example, this input representation is composed as follows: The first input channel contains a local $(k \times k)$ crop of the value prior $\widehat{V}_p^{z}$, centered on the current HL state z. A local map crop $\widehat{\Phi}^z$ forms the second channel. Additional channels contain the continuous state information for refinement, broadcasting the numerical value of a state component (for example a velocity or orientation) across all entries of the corresponding channel. During data collection, the layouts Φ_e per episode are stored with the dataset $\mathcal{D}$.

In contrast to the original work, we employ a SoftMax output layer instead of an argmax to the refined values. This way, the "Soft-VRN" outputs probabilities for all HL actions o, which allows computing likelihoods of HL state-action pairs.

3.4 Problem Statement

In this work, we investigate the problem of learning a policy in an offline setting, using a dataset $\mathcal{D}$, such that it performs well on a distribution $\mathbb{M}$ of goal-based MDPs. Each MDP m has the same $\mathcal{S}$ and $\mathcal{A}$ but may have different dynamics $\mathcal{P}_m$ (e.g. due to varying environment layout) as well as start and goal state distributions $\mathcal{S}_{0,m}$ and $\mathcal{S}_{g,m}$. The data $\mathcal{D}$ may only be collected on some of the MDPs and limited start-goal combinations. In summary, the objective is to learn a (hierarchical) policy that maximizes returns across all MDPs as well as starts and goals:

$$\max_{\omega,\pi} \mathbb{E}_{m\sim\mathbb{M}, s_0\sim\mathcal{S}_{0,m}, g\sim\mathcal{S}_{g,m}, \mathcal{P}_m} \left[\sum_{t=0}^{T-1} \gamma^t r\left(s_{t+1}, g\right)\right]. \tag{2}$$

The key challenge lies in inferring the latent HL actions o of the HL policy ω. Since they are the interface between the HL and LL policies, they need to be aligned between both for good performance.

4 HORIBLe-VRN Offline Learning Algorithm

This section presents our two-stage hierarchical offline learning scheme. We first take a look at the graphical model of our hierarchical policy in Sect. 4.1. Then Sect. 4.2 describes the pre-processing of the collected data. Sect. 4.3 presents the hierarchical IL stage of our algorithm, which is followed up by an offline RL refinement stage for the HL policy as described in Sect. 4.4. An overview over our algorithm HORIBLe-VRN is presented in Algorithm 2, at the end of the paper.

4.1 Graphical Model

We first need to understand how our hierarchical policy would have generated the data $\mathcal{D}$. Figure 2 depicts an exemplary state-action sequence, where LL states s_t and LL actions a_t are observed in the data. Please note the temporal abstraction, where the h_i are the time stamps when new HL actions o_{h_i} are selected based on the VRN inputs $z^{\mathcal{I}}_{h_i}$ at that time. As the VRN inputs deterministically depend on observed quantities like the state s_{h_i}, goal state g_e, and layout Φ_e, they are also marked observed. In the example, $h_0 = 0$ and $h_1 = 2$. This means that the first (latent) HL action o_{h_0}, selected based on $z^{\mathcal{I}}_{h_0}$, results in two LL transitions.

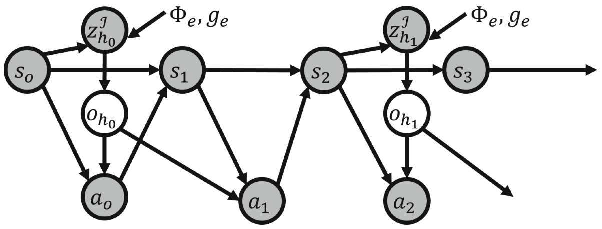


Fig. 2. Graphical model of the data generation process with our hierarchical policy architecture. Gray variables are observed from the data $\mathcal{D}$ or can be readily inferred from them, whereas white circles represent latent variables. (Color figure online)

Based on the graphical model, we can derive the probability of a latent HL action o_{h_i} in the sequence taking a specific value o given the data $\mathcal{D}$ as follows:

$$p\left(o_{h_i} = o|\mathcal{D}\right) = p\left(o_{h_i} = o|\mathcal{D}_{h_i}\right) = \frac{p\left(o_{h_i} = o|z^{\mathcal{I}}_{h_i}\right) \prod_{t\in[h_i,h_{i+1}-1]} p\left(a_t|s_t, o\right)}{\sum_{o'\in\mathcal{O}} \left(p\left(o_{h_i} = o'|z^{\mathcal{I}}_{h_i}\right) \prod_{t\in[h_i,h_{i+1}-1]} p\left(a_t|s_t, o'\right)\right)}. \tag{3}$$

We notice from Fig. 2 that HL action o_{h_i} only affects the LL transitions from h_i to h_{i+1}, until the next HL action is selected in time step h_{i+1}. The probabilities can be replaced by the (hypothetically) data generating policies $\omega^{\mathcal{D}}$ and $\pi^{\mathcal{D}}$ with parameters $\phi^{\mathcal{D}}$ and $\theta^{\mathcal{D}}$, respectively. For improved numerical stability of our implementation, we furthermore apply an $\exp\left(\log\left(...\right)\right)$ transform, which results in

$$p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \phi^{\mathcal{D}}, \theta^{\mathcal{D}}\right) = \exp\left(\log \omega^{\mathcal{D}}_{\phi^{\mathcal{D}}} + \sum_{t\in[...]} \log \pi^{\mathcal{D}}_{\theta^{\mathcal{D}}} - \log\left(\sum_{o'\in\mathcal{O}} \exp\left(\log \omega^{\mathcal{D}}_{\phi^{\mathcal{D}}} + \sum_{t\in[...]} \log \pi^{\mathcal{D}}_{\theta^{\mathcal{D}}}\right)\right)\right), \tag{4}$$

dropping some notation for improved readability.

4.2 Dataset Pre-Processing

In a first pre-processing step, we need to determine the time stamps h_i, forming the set $\mathcal{H}$, at which new HL actions are selected. For that, we make use of our assumptions on the HL action selection process (Sect. 3.2): New o are selected when the HL state z changes, the sub-episode horizon H runs out, or the episode terminates (episode termination flag $d = 1$). See Algorithm 1.

Algorithm 1: Computation of HL time stamps

```
1 $t \leftarrow 0, k \leftarrow 0, h_0 \leftarrow 0, \mathcal{H} \leftarrow \{h_0\}, i \leftarrow 1$
2 while $t < |\mathcal{D}|$ do
3     if $f_{\mathcal{Z}}\left(s_{h_{i-1}+k}\right) \neq f_{\mathcal{Z}}\left(s_{h_{i-1}}\right)$ then $h_i \leftarrow h_{i-1} + k$        // HL state change
4     else if $d_{h_{i-1}+k}$ then $h_i \leftarrow h_{i-1} + k + 1$                  // episode end
5     else if $k = H - 1$ then $h_i \leftarrow h_{i-1} + H$                   // sub-episode end
6     if $h_i$ was assigned above then $\mathcal{H} \leftarrow \mathcal{H} \cup \{h_i\}$, $i \leftarrow i+1$, $k \leftarrow 0$, and $t \leftarrow t+1$
7     else $k \leftarrow k + 1$ and $t \leftarrow t + 1$
8 end
9 return $\mathcal{H}$
```

In the next step, the VRN inputs $z^{\mathcal{I}}_{h_i}$ at the time stamps h_i are determined. For that, first, for every episode e in the data the prior (shortest path) plan $\widetilde{V}^{\mathcal{Z}}_{p,e}$ is determined via value iteration (VI) using layout Φ_e and goal state g_e from the data, as detailed in [38]. Finally, the accumulated HL rewards within a sub-episode and corresponding accumulated discount factors are calculated as

$$r^{\mathcal{Z}}_{h_i} = f_{r_{\mathcal{Z}}}\left(h_i, \mathcal{D}\right) = \sum_{t \in [h_i : h_{i+1}-1]} \gamma^{t-h_i} r_t \quad \text{and} \quad \gamma^{\mathcal{Z}}_{h_i} = \gamma^{h_{i+1}-h_i}. \tag{5}$$

As a result, the dataset $\mathcal{D}$ will have the following form after the pre-processing

$$\mathcal{D} = \left(\mathcal{D}_{h_i}\right)_{h_i \in \mathcal{H}} = \left(\left(z^{\mathcal{I}}_{h_i}, r^{\mathcal{Z}}_{h_i}, \gamma^{\mathcal{Z}}_{h_i}, d_{h_i}, z^{\mathcal{I}}_{h_i+1}, \left((s_t, a_t, s_{t+1})\right)_{t=h_i}^{h_i-1}\right)\right)_{h_i \in \mathcal{H}} \tag{6}$$

containing tuples $\mathcal{D}_{h_i}$ of HL VRN input transitions and corresponding sub-sequences of LL state-action transitions.

4.3 Stage 1: Hierarchical IL via EM Procedure

As a first step, we want to imitate the data $\mathcal{D}$ by finding parameters $\Theta^{\mathcal{D}} = \{\phi^{\mathcal{D}}, \theta^{\mathcal{D}}\}$ that maximize the likelihood. Since the HL action sequence $\boldsymbol{o} = \{o_{h_0}, o_{h_1}, \ldots\}$ is unknown, we employ an iterative Expectation-Maximization (EM) [8] scheme.

Expectation (E-Step). Following the EM scheme, we establish a lower bound

$$\begin{aligned}
&\mathcal{Q}\left(\Theta^{\mathcal{D}}|\Theta^{\text{old}}\right) = \mathbb{E}_{\boldsymbol{o}|\mathcal{D},\Theta^{\text{old}}} \log p\left(\mathcal{D}, \boldsymbol{o}|\Theta^{\mathcal{D}}\right) \\
&= \mathbb{E}_{\boldsymbol{o}|\mathcal{D},\Theta^{\text{old}}} \sum_{i\in[0,|\mathcal{H}|-1]} \left(\log \omega^{\mathcal{D}}_{\phi^{\mathcal{D}}}\left(o_{h_i}|z^{\mathcal{I}}_{h_i}\right) + \sum_{t\in[h_i,h_{i+1}-1]} \log \pi^{\mathcal{D}}_{\theta^{\mathcal{D}}}\left(a_t|s_t,o\right)\right) \\
&\qquad = \sum_{i\in[0,|\mathcal{H}|-1]} \sum_{o\in\mathcal{O}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\text{old}}\right) \log \omega^{\mathcal{D}}_{\phi^{\mathcal{D}}}\left(o|z^{\mathcal{I}}_{h_i}\right) + \\
&\qquad\quad \sum_{i\in[0,|\mathcal{H}|-1]} \sum_{o\in\mathcal{O}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\text{old}}\right) \sum_{t\in[h_i,h_{i+1}-1]} \pi^{\mathcal{D}}_{\theta^{\mathcal{D}}}\left(a_t|s_t,o\right) \quad (7)
\end{aligned}$$

where the $p\left(o_{h_i} = o|\mathcal{D}, \Theta^{\text{old}}\right)$ are obtained using Eq. (4) with the "old" parameters Θ^{old}.

Maximization (M-Step). In the maximization step, we obtain new model parameters Θ^{new}. Since the terms for the HL and LL parameters are separated from each other in Eq. (7), we can optimize these parameters separately with

$$\phi^{\mathcal{D},\text{new}} = \arg\max_{\phi^{\mathcal{D}}} \mathbb{E}_{\boldsymbol{o}|\mathcal{D},\Theta^{\text{old}}} \log p\left(\mathcal{D}, \boldsymbol{o}|\phi^{\mathcal{D}}\right) \quad (8)$$

and

$$\theta^{\mathcal{D},\text{new}} = \arg\max_{\theta^{\mathcal{D}}} \mathbb{E}_{\boldsymbol{o}|\mathcal{D},\Theta^{\text{old}}} \log p\left(\mathcal{D}, \boldsymbol{o}|\theta^{\mathcal{D}}\right). \quad (9)$$

We approximate the M-step, similar to [30], as a single gradient step, utilizing the Adam [21] optimizer. See Algorithm 2 for the parameter update formulas.

4.4 Stage 2: Offline RL HL Policy Refinement

In case the data $\mathcal{D}$ was collected by some non-optimal policy, the algorithm in the previous stage learns to imitate sub-optimal behavior. Therefore, we propose an offline RL refinement stage for the HL VRN policy, based on Batch-Constrained deep Q-learning (BCQ) [13]. BCQ is an offline DQN algorithm that at the core learns a state-action value function Q. Simultaneously, a state-conditioned generative model $G : \mathcal{S} \rightarrow \mathcal{A}$ of the data is learned, which is used to exclude actions from the action selection, which are unlikely under the data generating policy.

Our modified BCQ algorithm operates in the $(\mathcal{Z}, \mathcal{O}, H)$ HL problem abstraction. The generative model is readily available from *stage 1* as $G := \omega_{\phi^{\mathcal{D}}}\left(o|z^{\mathcal{I}}\right)$ and, therefore, does not need to be learned separately, which is our first modification. As a result, the constrained action selection can be written as

$$o = \arg \max_{\tilde{o}\in\tilde{\mathcal{O}}(z^{\mathcal{I}})} Q_{\phi_{(\text{target})}}\left(z^{\mathcal{I}}, \tilde{o}\right) \quad (10)$$

with

$$\tilde{\mathcal{O}}\left(z^{\mathcal{I}}\right) = \left\{ o \in \mathcal{O} \left| \frac{\omega_{\phi^{\mathcal{D}}}\left(o|z^{\mathcal{I}}\right)}{\max_{\hat{o}} \omega_{\phi^{\mathcal{D}}}\left(\hat{o}|z^{\mathcal{I}}\right)} > \tau_{\text{BCQ}} \right.\right\}. \quad (11)$$

Our second modification affects the Q-function learning. While the VRN inputs $z^{\mathcal{I}}_{h_i}$ and $z^{\mathcal{I}}_{h_{i+1}}$ are part of the HL transition tuples $\mathcal{D}_{h_i}$ in our pre-processed dataset, the corresponding (latent) HL actions o_{h_i} are, again, unknown. Therefore, we re-write the Q-learning update replacing the latent o_{h_i} by their expectation with respect to the data generating policy parameters $\Theta^{\mathcal{D}}$, making use of our ability to compute $p\left(o_{h_i} = o|\mathcal{D}_{h_i}\right)$ according to Eq. (4), resulting in

$$\begin{aligned}&\arg\min_{\phi} \mathbb{E}_{z^{\mathcal{I}}_{h_i}, o_{h_i}, r^{\mathcal{Z}}_{h_i}, \gamma^{\mathcal{Z}}_{h_i}, d_{h_i}, z^{\mathcal{I}}_{h_{i+1}}|\mathcal{D}} \left[\mathcal{L}_{\text{TD}}\left(h_i, o = o_{h_i}|\phi, \phi_{\text{target}}\right)\right] = \\ &\arg\min_{\phi} \mathbb{E}_{z^{\mathcal{I}}_{h_i}, r^{\mathcal{Z}}_{h_i}, \gamma^{\mathcal{Z}}_{h_i}, d_{h_i}, z^{\mathcal{I}}_{h_{i+1}}|\mathcal{D}} \left[\sum_{o \in \mathcal{O}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\mathcal{D}}\right) \mathcal{L}_{\text{TD}}\left(h_i, o|\phi, \phi_{\text{target}}\right)\right]\end{aligned} \tag{12}$$

with the temporal-difference (TD) loss for the Q-function (network) denoting as

$$\begin{aligned}&\mathcal{L}_{\text{TD}}\left(h_i, o|\phi, \phi_{\text{target}}\right) = \\ &\qquad l_\kappa\left(r^{\mathcal{Z}}_{h_i} + \left(1 - d_{h_i}\right) \gamma^{\mathcal{Z}}_{h_i} Q_{\phi_{\text{target}}}\left(z^{\mathcal{I}}_{h_{i+1}}, o'\right) - Q_\phi\left(z^{\mathcal{I}}_{h_i}, o\right)\right)\end{aligned} \tag{13}$$

with

$$o' = \arg \max_{\tilde{o}' \in \tilde{\mathcal{O}}\left(z^{\mathcal{I}}_{h_{i+1}}\right)} Q_\phi\left(z^{\mathcal{I}}_{h_{i+1}}, \tilde{o}'\right) \tag{14}$$

and l_κ being the Huber loss [18].

5 Empirical Evaluation

We will now empirically evaluate HORIBLe-VRN on a long-horizon agent navigation task, investigating the research hypotheses *H.1-3* from the introduction.

5.1 Environment

To investigate our research hypotheses, we need an environment with certain properties: Inherent task hierarchy, multiple layouts with different starts and goals requiring generalization, and agent dynamics that make simple HL shortest path planning insufficient. As a result, we implement, using MuJoCo [33], a point mass agent with an orientation influencing the motion that navigates different environment layouts. An example layout is shown in the bottom left of Fig. 1.

The continuous 6D state space $\mathcal{S}$ consists of the x and y position, orientation ϑ, as well as the corresponding velocities $(\dot{x}, \dot{y}, \dot{\vartheta})$. The action space consists of a linear acceleration a_v in orientation direction and the rotational acceleration $a_\vartheta = \ddot{\vartheta}$. The time horizon is $T = 500$ and the sparse reward signal results in -1 per step until the goal is reached (0). For the 2D (HL) abstraction $\mathcal{Z}$, the environment is tiled into 25×25 cells with $\bar{x}$ and $\bar{y}$ index. The HL VRN policy receives a $6 \times 7 \times 7$ input consisting of 7×7 (binary occupancy) layout and shortest path prior plan (value map) crops as well as $\vartheta, \dot{x}, \dot{y}, \dot{\vartheta}$ broadcasted across the remaining four channels, respectively (see Sect. 3.3 and [38]). The HL action space $\mathcal{O}$ contains four HL actions o corresponding to the sub-task of moving to the neighboring HL state (tile) z in direction 'North', 'East', 'South', or 'West', respectively. To address these sub-tasks, the low-level MLP policy receives as input the state s alongside the tile representing the sub-goal, and an 8D binary occupancy vector for the neighboring tiles. The sub-goal horizon is $H = 10$.

Regarding the layouts as well as starts and goals, we distinguish 3 settings:

1. We have a first set of 12 layouts (also used for data collection) with 25 different pairs of start and goal tiles, each. The continuous start s_0 arises by uniformly sampling x and y from the start tile alongside an initial orientation $\in [-\pi, \pi]$.
2. To investigate generalization towards other starts and goals, we additionally allow start tiles being goal tiles and vice versa for the aforementioned setting.
3. To investigate generalization towards other layouts, we have a second set of 12 different layouts with 50 pairs of start and goal tiles, each.

5.2 Data Collection

For the purpose of data collection, we, first, train a hierarchical policy similar to the one described in Sect. 3.2 online, interacting with the environment (first layout setting): The HL VRN is trained (as in [38]) via double DQN [34] with HER [1] (featuring argmax action selection). The LL MLP policy is trained via SAC [17]. We use a hierarchical architecture for data collection because no flat policy was able to solve the multi layout navigation. Furthermore, this ensures that our HORIBLe-VRN conforms to the assumptions about the generative process. Using the obtained policy weights, we collect two different sets of data:

1. An "optimal" set of data, using the weights of the converged policy, which achieved 100 % success in reaching the goal tiles.
2. A "mediocre" dataset, using weights of an intermediate policy, which only achieved 70 % successful goal-reaching.

For each of these, we collect 20 rollouts for each start-goal tile combination (25) for each of the 12 layouts of the first layout setting (see previous section).

5.3 Baselines

Flat Offline Learning Baselines. We compare to two different flat behavioral cloning [3] baselines: The first (BC) uses the MLP architecture used as LL policy in our approach. The second (BC VRN) uses the VRN architecture used as HL policy in our approach, with two small modifications: 1) The number and size of the fully-connected layers is increased to match the LL MLP policy of our approach. 2) The SoftMax layer is replaced by a linear layer to output means and standard deviations for the 2D continuous actions. Besides that, we train the MLP architecture via TD3-BC [14] offline RL (ORL), as an additional baseline.

Heuristic Hierarchical Imitation Learning (HHIM). We compare the first stage of our approach (Sect. 4.3) to a hierarchical IL baseline, similar to h-BC [24]. We, however, first, need to infer the values of the latent HL actions o_{h_i}. For this, we use a simple heuristic that makes use of the (semantic) meaning the HL actions have in our navigation task: Requesting a HL state change in a certain direction (e.g. 'North') from the LL policy, which may or may not succeed at this sub-task. For this simple baseline, we assume that the HL state change from z_{h_i} to $z_{h_{i+1}}$, seen in the data $\mathcal{D}$, is intentional, and choose o_{h_i} accordingly. This may decently hold true for data collected with optimal policies. Using these approximate HL actions $\bar{o}_{h_i}$ allows to individually perform BC for both, the HL and the LL policy denoting as (with f_{h_i} returning the HL time stamp corresponding to t)

$$\max_{\phi} \mathbb{E}_{z^{\mathcal{I}}_{h_i}, \bar{o}_{h_i} | \mathcal{D}} [\log \omega_{\phi} \left(\bar{o}_{h_i} | z^{\mathcal{I}}_{h_i} \right)] \tag{15}$$

and

$$\max_{\theta} \mathbb{E}_{s_t, a_t, \bar{o}_{f_{h_i}(t)} | \mathcal{D}} \left[\log \pi_{\theta} \left(a_t | s_t, \bar{o}_{f_{h_i}(t)} \right) \right]. \tag{16}$$

Heuristic HL BCQ Baseline (HBCQ). We compare the second (HL refinement) stage of our approach (Sect. 4.4) to directly learning the HL policy via BCQ [13] offline RL, without using $\omega_{\phi^{\mathcal{D}}} \left(o | z^{\mathcal{I}} \right)$ as generative model. Again, we first need to infer the values of the latent HL actions o_{h_i}, using the same heuristic as for HHIM. With these approximate HL actions $\bar{o}_{h_i}$, Eq. 12 denotes as

$$\arg\min_{\phi} \mathbb{E}_{z^{\mathcal{I}}_{h_i}, \bar{o}_{h_i}, r^{\mathcal{Z}}_{h_i}, \gamma^{\mathcal{Z}}_{h_i}, d_{h_i}, z^{\mathcal{I}}_{h_{i+1}} | \mathcal{D}} \left[\mathcal{L}_{\text{TD}} \left(h_i, \bar{o}_{h_i} | \phi, \phi_{\text{target}} \right) \right], \tag{17}$$

without the necessity of doing an expectation over all possible HL actions o. We use the same LL policy $\pi_{\theta^{\mathcal{D}}}$, from the previous step, as for our approach.

5.4 Results and Discussion

The results of evaluating the offline trained policies in the environment are presented in Table 1. We report results for both datasets ("optimal", "mediocre") and all three settings of layouts and starts/goals (as detailed in 5.1), each. The numerical values in the table reflect success rates of reaching the goal tile, reported as mean ± standard error across the best performing policies of 10 individual runs (seeds) of the respective algorithm. A success rate is determined by performing 10 rollouts each, with sampled start-goal pairs, for any of the 12 layouts. In the following, we will interpret these results in the light of our research hypotheses.

H.1: **Using a Planning-Based Policy Containing a VRN Improves Generalization in the Offline Learning Setting.** We, first, focus the discussion of the results on the (stage 1) imitation learning (IL) setting. Comparing the *flat* baselines BC, ORL, and BC VRN, it turns out that BC VRN incorporating

Table 1. Mean ± standard error across 10 seeds of success rates of best policies.

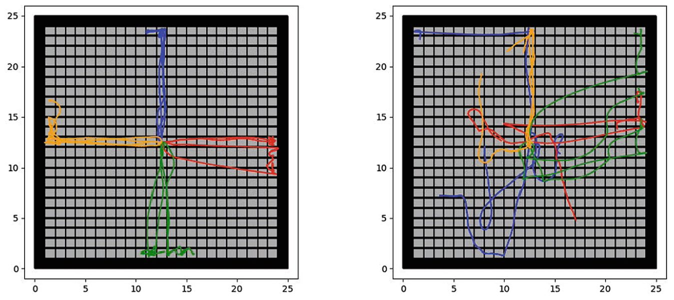
	Optimal Data			Mediocre Data		
	1) collect	2) add. s_0/g	3) other lay.	1) collect	2) add. s_0/g	3) other lay.
BC	0.744 ± 0.006	0.614 ± 0.010	0.468 ± 0.006	0.457 ± 0.009	0.370 ± 0.005	0.270 ± 0.006
BC VRN	0.984 ± 0.003	0.880 ± 0.014	0.877 ± 0.012	0.633 ± 0.007	0.478 ± 0.012	0.422 ± 0.012
ORL	0.740 ± 0.009	0.602 ± 0.006	0.478 ± 0.011	0.443 ± 0.009	0.348 ± 0.011	0.203 ± 0.006
HHIM	0.744 ± 0.011	0.647 ± 0.014	0.638 ± 0.015	0.501 ± 0.007	0.419 ± 0.012	0.391 ± 0.009
Stage 1 (Sect. 4.3)	0.564 ± 0.019	0.428 ± 0.019	0.389 ± 0.017	0.520 ± 0.019	0.394 ± 0.015	0.344 ± 0.019
Stage 1, w init	**1.000 ± 0.000**	0.945 ± 0.004	0.959 ± 0.004	0.746 ± 0.003	0.665 ± 0.004	0.630 ± 0.008
HORIBLe-VRN(Ours)	**1.000 ± 0.000**	**0.963 ± 0.002**	**0.983 ± 0.002**	**0.808 ± 0.003**	**0.744 ± 0.005**	**0.717 ± 0.010**
HBCQ	0.439 ± 0.014	0.373 ± 0.007	0.370 ± 0.013	0.413 ± 0.008	0.323 ± 0.012	0.286 ± 0.009

planning clearly performs best while also generalizing better, dropping the least in success rate on the (unseen) other layouts 3). So, integrating planning in the form of a VRN into the policy improves performance and generalization.

H.2: **Learning a Hierarchical Policy with Temporal Abstraction from Offline Data Can Further Improve Performance.** With our stage 1 *hierarchical* IL scheme from Sect. 4.3 we add the aspects of hierarchy and temporal abstraction compared to the also planning-based BC VRN. Using the EM procedure from Sect. 4.3 for the necessary inference of unknown, latent HL actions, our stage 1 (with HHIM weight initialization) clearly outperforms the hierarchical HHIM baseline of similar architecture that only uses heuristic HL action inference[1]. Trained with optimal data, our stage 1 (w init) recovers the performance of the data collecting policy on the corresponding layouts 1). Incorporating the VRN planning, it furthermore generalizes well to different starts, goals, and layouts. Compared to the *flat* BC VRN, our stage 1 (w init) generalizes better to the settings 2) and 3), especially in case of the mediocre dataset, where it performs 0.266 to 0.295 better, despite the necessary HL action (o) inference.

[1] This is not due to additional optimization steps for initialization. The success rates of our approach and HHIM stop to significantly improve long before $N_{\text{iter}} = 10000$.

Effect of Initialization. For our stage 1 hierarchical IL from Sect. 4.3 to perform well, initializing the weights Θ is crucial. We do this by first running HHIM for $N_{\text{iter}} = 2500$[2]. Afterwards, we initialize the first stage of our algorithm (Sect. 4.3) with these weights and run it for $N_{\text{iter}} = 10000$. Figure 3 shows the effect of the initialization on the (LL) policy behavior. We evaluate how it behaves for different HL actions o, in an empty environment, starting in the middle. With the initialization (Fig. 3a), the agent moves "North", "East", ... when the corresponding o is selected. Without initialization (Fig. 3b), the stage 1 hierarchical IL from Sect. 4.3 can freely assign behaviors to the various o, which not only leads to different permutations ($o = 3$: "North" instead of "West"), but also to not any o being clearly assigned to "West", in the example. The reasonable o assignment in Fig. 3a might be explained by the weight initialization acting as a prior on the assignment. The HHIM algorithm used for initialization assigns an o to its semantic meaning as a result of the heuristic rules used for the o-inference.



(a) With HHIM init. (b) Without init.

Fig. 3. LL policy behavior in empty maze for HL action $o \in \{0 : \text{"North"}, 1 : \text{"East"}, 2 : \text{"South"}, 3 : \text{"West"}\}$; colored blue, red, green, and orange, respectively. We did 4 rollouts with initial orientation $\vartheta \in \{0, \frac{\pi}{2}, \pi, -\frac{\pi}{2}\}$, for each o. (Color figure online)

H.3: **Adding our Offline RL Refinement Stage for the HL Policy Improves Performance/Robustness When the Data is Collected by Sub-Optimal Policies.** Our full HORIBLe-VRN algorithm adds our (stage 2) modified BCQ offline RL refinement (Sect. 4.4) for the HL VRN on top of the hierarchical IL (stage 1). The results in Table 1 show a further success rate improvement, especially for the mediocre dataset, where the imitation learned policy is not already close to optimal. The HBCQ baseline that learns the HL policy via offline RL from scratch, using heuristically inferred o-s, performs clearly worse.

[2] This is less than for a full training ($N_{\text{iter}} = 10000$ for BC (VRN), HHIM, stage 1).

Algorithm 2: Our HORIBLe-VRN Algorithm

```
   Input  : Dataset $\mathcal{D}$, initial parameters $\Theta^{\text{init}} = \{\phi^{\text{init}}, \theta^{\text{init}}\}$
   Output: Optimized policy parameters $\Theta^{\text{final}} = \{\phi^{\text{final}}, \theta^{\text{final}}\}$
   /* Stage 0: Dataset pre-processing as outlined in Sect.4.2 */
   /* Stage 0.5: Optionally obtain initial parameters $\Theta^{\text{init}}$ using the
      HHIM algorithm described in Sect. 5.3 */
   /* Stage 1: Hierarchical Imitation Learning (Sect.4.3) */
 1 $\Theta^{\text{old}} \leftarrow \Theta^{\text{init}}$, $\Theta^{\mathcal{D}} \leftarrow \Theta^{\text{init}}$
 2 for $n = 1 : N_{iter}$ do
 3     Sample batch $\mathcal{B}$ of time stamps $h_i$ ($\neq |\mathcal{H}|$), obtain tuples $\mathcal{D}_{h_i}$, and calculate,
       using Eq. (4), $p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\text{old}}\right)$ $\forall o \in \mathcal{O}$ and $\forall h_i \in \mathcal{B}$.
 4     if $(n \mod k_{HL}) = 0$ then
           /* HL param. gradient step to optim. Eq. (8) on $\mathcal{B}$ */
 5         $\phi^{\mathcal{D}} \leftarrow \phi^{\mathcal{D}} + \alpha_{\text{HL}} \nabla_\phi \sum_{o\in\mathcal{O}} \sum_{h_i\in\mathcal{B}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\text{old}}\right) \log \omega_\phi \left(o|z^{\mathcal{I}}_{h_i}\right)$
 6     end
 7     if $(n \mod k_{LL}) = 0$ then
           /* LL param. gradient step to optim. Eq. (9) on $\mathcal{B}$: */
 8         $\pi^{\mathcal{D}} \leftarrow$
           $\pi^{\mathcal{D}} + \alpha_{\text{LL}} \nabla_\theta \sum_{o\in\mathcal{O}} \sum_{h_i\in\mathcal{B}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\text{old}}\right) \sum_{t\in[h_i, h_{i+1}-1]} \log \pi_\theta \left(a_t|s_t, o\right)$
 9     end
10     $\Theta^{\text{old}} \leftarrow \Theta^{\mathcal{D}}$
11 end
12 $\phi = \phi_{\text{target}} \leftarrow \phi^{\mathcal{D}}, \theta^{\text{final}} \leftarrow \theta^{\mathcal{D}}$
   /* Stage 2: Offline RL HL Policy Refinement (Sect.4.4) */
13 Calculate, using Eq. (4), $p\left(o_{h_i} = o|\mathcal{D}_{h_i}, \Theta^{\mathcal{D}}\right)$ $\forall h_i \in \mathcal{H} \setminus |\mathcal{H}|$ and $\forall o \in \mathcal{O}$.
14 for $m = 1 : M_{iter}$ do
15     Sample batch $\mathcal{B}$ of HL time stamps $h_i$ ($\neq |\mathcal{H}|$)
       /* Gradient step to optim. Eq. (12) on $\mathcal{B}$; $\mathcal{L}_{\text{TD}}$: Eq. (13) */
16     $\phi \leftarrow \phi - \alpha_{\text{BCQ}} \nabla_\phi \left[\sum_{o\in\mathcal{O}} p\left(o_{h_i} = o|\mathcal{D}_{h_i}\right) \mathcal{L}_{\text{TD}}\left(o_{h_i} = o, h_i|\phi, \phi_{\text{target}}\right)\right]$
17     $\phi_{\text{target}} \leftarrow \tau \cdot \phi + (1-\tau) \cdot \phi_{\text{target}}$        // Polyak target network update
18 end
19 $\phi^{\text{final}} \leftarrow \phi$
20 return $\Theta^{final} = \{\phi^{final}, \theta^{final}\}$
```

6 Conclusion

In this work, we propose HORIBLe-VRN, a two-stage algorithm to learn a hierarchical policy architecture containing a top-level Value Refinement Network purely from (offline) demonstration data: First, a hierarchical imitation learning stage incorporates inference of the unknown, latent HL actions and policy updates into an iterative EM-style algorithm tailored to our architecture. Subsequently, we use our proposed modified version of the offline RL BCQ algorithm to further refine the HL VRN policy. We empirically investigated several research hypotheses in long horizon, sparse reward agent navigation finding out that the planning-based VRN policy enables generalization (*H.1*), also in the offline setting, the hierarchical policy architecture with temporal abstraction further improves performance (*H.2*), and that our HL policy refinement improves robustness when demonstrations originate from sub-optimal policies (*H.3*).

An interesting direction for future work could be to scale up the approach to more complex problem settings by, for example, allowing for more flexible HL state space abstractions or even learning them as well.

Ethical Statement. We did not collect or process any personal data, for this work. All data was collected using a physics simulation of a point mass agent. There are several possible future applications for our research, like, for example, in autonomous vehicles or robotics, which hopefully have a positive impact on society. There are, however, also risks of negative societal impact, through the form of application itself, the impact on the job market, or real-world application without proper verification and validation. Such factors should be taken into consideration when designing applications.

References

1. Andrychowicz, M., et al.: Hindsight experience replay. In: Advances in Neural Information Processing Systems (NeurIPS), pp. 5048–5058 (2017)
2. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: AAAI Conference on Artificial Intelligence (2017)
3. Bain, M., Sammut, C.: A framework for behavioural cloning. In: Machine Intelligence, vol. 15, pp. 103–129 (1995)
4. Baum, L.E.: An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. Inequalities **3**(1), 1–8 (1972)
5. Christen, S., Jendele, L., Aksan, E., Hilliges, O.: Learning functionally decomposed hierarchies for continuous control tasks with path planning. IEEE Robot. Autom. Lett. **6**(2), 3623–3630 (2021)
6. Daniel, C., Van Hoof, H., Peters, J., Neumann, G.: Probabilistic inference for determining options in reinforcement learning. Mach. Learn. **104**, 337–357 (2016)
7. Dayan, P., Hinton, G.E.: Feudal reinforcement learning. Adv. Neural Inf. Process. Syst. (NeurIPS) **5**, 271–278 (1992)
8. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Stat. Soc.: Ser. B (Methodological) **39**(1), 1–22 (1977)
9. Eysenbach, B., Salakhutdinov, R., Levine, S.: Search on the replay buffer: bridging planning and reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 32 (2019)
10. Fox, R., Krishnan, S., Stoica, I., Goldberg, K.: Multi-level discovery of deep options. arXiv preprint arXiv:1703.08294 (2017)
11. Fox, R., et al.: Hierarchical variational imitation learning of control programs. arXiv preprint arXiv:1912.12612 (2019)
12. Francis, A., et al.: Long-range indoor navigation with PRM-RL. IEEE Trans. Robot. (2020)
13. Fujimoto, S., Conti, E., Ghavamzadeh, M., Pineau, J.: Benchmarking batch deep reinforcement learning algorithms. arXiv preprint arXiv:1910.01708 (2019)

14. Fujimoto, S., Gu, S.: A minimalist approach to offline reinforcement learning. Adv. Neural Inf. Process. Syst. (NeurIPS) **34**, 20132–20145 (2021)
15. Giammarino, V., Paschalidis, I.: Online Baum-Welch algorithm for hierarchical imitation learning. In: Conference on Decision and Control (CDC), pp. 3717–3722. IEEE (2021)
16. Gieselmann, R., Pokorny, F.T.: Planning-augmented hierarchical reinforcement learning. IEEE Robot. Autom. Lett. **6**(3), 5097–5104 (2021)
17. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning (ICML), pp. 1861–1870 (2018)
18. Huber, P.J.: Robust estimation of a location parameter. Ann. Math. Stat., 73–101 (1964)
19. Hutsebaut-Buysse, M., Mets, K., Latré, S.: Hierarchical reinforcement learning: a survey and open research challenges. Mach. Learn. Knowl. Extract. **4**(1), 172–221 (2022)
20. Jing, M., et al.: Adversarial option-aware hierarchical imitation learning. In: International Conference on Machine Learning (ICML), pp. 5097–5106 (2021)
21. Kingma, D.P., Ba, J.: ADAM: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
22. Krishnan, S., Fox, R., Stoica, I., Goldberg, K.: DDCO: discovery of deep continuous options for robot learning from demonstrations. In: Conference on Robot Learning (CoRL), pp. 418–437 (2017)
23. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative Q-learning for offline reinforcement learning. Adv. Neural Inf. Process. Syst. (NeurIPS) **33**, 1179–1191 (2020)
24. Le, H., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., Daumé III, H.: Hierarchical imitation and reinforcement learning. In: International Conference on Machine Learning (ICML), pp. 2917–2926 (2018)
25. Levy, A., Konidaris, G., Platt, R., Saenko, K.: Learning multi-level hierarchies with hindsight. In: International Conference on Learning Representations (ICLR) (2019)
26. Li, B., Li, J., Lu, T., Cai, Y., Wang, S.: Hierarchical learning from demonstrations for long-horizon tasks. In: International Conference on Robotics and Automation (ICRA), pp. 4545–4551. IEEE (2021)
27. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
28. Nardelli, N., Synnaeve, G., Lin, Z., Kohli, P., Torr, P.H., Usunier, N.: Value propagation networks. In: International Conference on Learning Representations (ICLR) (2019)
29. Prudencio, R.F., Maximo, M.R., Colombini, E.L.: A survey on offline reinforcement learning: taxonomy, review, and open problems. IEEE Trans. Neural Netw. Learn. Syst. (2023)
30. Salakhutdinov, R., Roweis, S.T., Ghahramani, Z.: Optimization with EM and expectation-conjugate-gradient. In: International Conference on Machine Learning (ICML), pp. 672–679 (2003)
31. Smith, M., Van Hoof, H., Pineau, J.: An inference-based policy gradient method for learning options. In: International Conference on Machine Learning (ICML), pp. 4703–4712. PMLR (2018)
32. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif. Intell. **112**(1–2), 181–211 (1999)

33. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: International Conference on Intelligent Robots and Systems, pp. 5026–5033. IEEE (2012)
34. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: AAAI Conference on Artificial Intelligence, vol. 30 (2016)
35. Vecerik, M., et al.: Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv:1707.08817 (2017)
36. Vezhnevets, A.S., et al.: FeUdal networks for hierarchical reinforcement learning. In: International Conference on Machine Learning (ICML), pp. 3540–3549 (2017)
37. Wöhlke, J., Schmitt, F., Van Hoof, H.: Hierarchies of planning and reinforcement learning for robot navigation. In: International Conference on Robotics and Automation (ICRA), pp. 10682–10688. IEEE (2021)
38. Wöhlke, J., Schmitt, F., Van Hoof, H.: Value refinement network (VRN). In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 3558–3565 (2022)
39. Zhang, Z., Paschalidis, I.: Provable hierarchical imitation learning via EM. In: International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 883–891 (2021)
40. Zheng, B., Verma, S., Zhou, J., Tsang, I.W., Chen, F.: Imitation learning: progress, taxonomies and challenges. IEEE Trans. Neural Netw. Learn. Syst., 1–16 (2022)