# Advances in Adaptive Skill Acquisition

Juraj Holas and Igor Farkaš

Faculty of Mathematics, Physics and Informatics*
Comenius University in Bratislava, Slovakia
{juraj.holas,igor.farkas}@fmph.uniba.sk

**Abstract.** Hierarchical Reinforcement Learning (HRL) represents a viable approach to learning complex tasks, especially those with an inner hierarchical structure. The HRL methods decompose the problem into a typically two-layered hierarchy. At the lower level, individual *skills* are created to solve specific non-trivial subtasks, such as locomotion primitives. The high-level agent can then use these skills as its actions, enabling it to tackle the overall task. The identification of an appropriate skill set, however, is a difficult problem by itself. Most current approaches solve it using a pre-training phase, in which skills are trained and fixed, before launching the training of the high-level agent. Having the skill set fixed prior to main training session can however impose flaws on the HRL system – especially if a useful skill was not successfully identified, and hence is missing from the skill set. Our *Adaptive Skill Acquisition* framework (ASA) aims specifically for these situations. It can be plugged onto existing HRL architectures and fix the defects within the pre-trained skill set. During the training of the high-level agent, ASA detects a missing skill, trains it, and integrates it into the existing system. In this paper, we present new improvements to the ASA framework, especially a new skill-training reward function, and support for skill-stopping functions enabling better integration. Furthermore, we extend our prior pilot tests into extensive experiments evaluating the functionality of ASA, in comparison to its theoretical boundaries. The source code of ASA is also available online[1].

**Keywords:** Hierarchical reinforcement learning · skill acquisition · adaptive model

## 1 Introduction

The field of Reinforcement Learning has been gaining significant attention within the past years, as being a viable pool of Machine Learning methods based on biologically motivated concepts. Thanks to its advancements, RL methods can easily surpass human-level performance in certain tasks. However, traditional ('flat') RL can still fall short if the problem at hand features several layers of abstraction.

* This research was supported by KEGA grant no. 042UK-4/2019

[1] https://github.com/holasjuraj/asa

To remedy these problems, the *Hierarchical Reinforcement Learning* (HRL) was introduced [24]. HRL approaches are designed to be able to abstract a hierarchical structure of the task, and reflect it within the architecture of the model. The HRL models hence contain *skills* – actions that are temporally extended in time, which are usually specialised to perform a specific subtask. A high-level agent operates on top of them to solve the main task, using the low-level skills to abstract the peculiarities of individual subtasks. This architecture hence creates an implicit or explicit leveled hierarchy within the model.

The process of acquiring skills is typically crucial to the model, and makes a core of individual HRL algorithms. Older methods like [20,24] specify and train the skill behaviors manually, which, however, comes with high risk of engineered bias. On the other hand, all modern approaches construct the skill set in automated or semi-automated way. The most used approach is to use a pre-training phase, during which the skills are trained using a surrogate reward [6,7,10,14,15,16]. After the pre-training phase has finished, the skills are fixed and training of the high-level agent starts.

However, this two-phased training of a HRL architecture comes with a problems stemming from the principle of *optimality under given hierarchy* [24]. If the pre-training phase fails to produce the perfect set of skills, the high-level agent will likely fail to find an optimal overall solution, as it is bounded by the suboptimal skill set. As an example, we can imagine a locomotion robot with a task to solve a maze, which is only given skills to move forward/backward and turn left, but no skill to turn right. The high-level agent, choosing from this limited skill set, may still find a solution how to navigate through the maze, but it will clearly be suboptimal for cases it should have turned right. This principle was also experimentally proven [12].

There has been a limited amount of research focused on training the whole hierarchy jointly, with the option to adapt all skills even during the high-level training [12,13,19]. However, most of them rely on using Universal-MDP which, despite its name, is a subset of MDP class suitable mostly for spatial 'reaching' tasks.

Addressing the problem of a missing skill, we introduced our framework of *Adaptive Skill Acquisition* (ASA) [8]. Its main purpose is to identify that a skill is missing during the training of the high-level agent, formulate and train the missing skill, and integrate it to the faulty skill set. ASA is designed as a pluggable component, which can be deployed onto almost any existing HRL architecture or those yet to come. In this paper we present the improvements that have been made to the model to make it even more robust, efficient, and reusable.

Additionally, all skills generated by previous approaches can be categorized into two disjoint types. The *subgoal-based* skill identification works by selecting a state with greater importance (subgoal), and then training a skill to reach this state from its proximity [2,10,12,14,17,19]. A typical subgoal example is a doorway in a grid-world environment. On the other hand, the *behavioral* skills are crafted to execute a specific behavior which is useful in any part of state-space, not only near a subgoal – such as walking for a legged robot [6,13,15]. To

the best of our knowledge, there has not been an approach which could train both *subgoal-based* and *behavioral* skills. In this work we experimentally confirm that ASA is able to train both types of skills, and it does so without the need of parameter change, or specifying which skill type it should create.

As a key contribution of this paper, we also present extended results of our method, compared to the previous pilot tests. We empirically demonstrate the successful performance of ASA on two distinctive environments, analyze the quality of the new skill, and compare our work to the HiPPO algorithm [13].

## 2 Preliminaries

We define a *Markov Decision Process* (MDP) as a tuple $\langle S, A, P, p_0, R, \gamma \rangle$, where $S \subseteq \mathbb{R}^{\dim(S)}$ is a set of states, $A$ is a set of actions, $P : S \times A \times S \to [0, 1]$ is a probability distribution describing the state transitions, $p_0 : S \to [0, 1]$ is a probability distribution of the initial state, $R : S \times A \to \mathbb{R}$ is a (possibly non-deterministic) reward function, $\gamma \in (0, 1]$ is a reward discount factor. Traditional ('flat') RL aims to find an optimal policy $\pi : S \times A \to [0, 1]$ that optimizes the expected overall discounted return $G(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where $\mathbb{E}_\pi$ denotes the expected value if an agent follows a policy $\pi$.

In Hierarchical RL, we do not optimize a single policy $\pi$, but rather a set of policies on two (or more) levels. We have a set of skills – low-level policies $\pi^L_1, \ldots, \pi^L_n$ that act using the original actions: $\pi^L_i(s_t) = a_t$. On top of them we have a manager – driven by a high-level policy $\pi^H$. Its purpose is to decide which skill will be used in a given situation, and thus its high-level actions $a^H_t$ are in fact invocations of skill policies: $\pi^H(s_t) = a^H_t \in \{\pi^L_1, \ldots, \pi^L_n\}$. The chosen skill then selects actions until the termination criterion is met, e.g. its time limit is reached. In our paper we study variable-length sequences of skill invocations, which we denote by $\delta = [a^H_t, \ldots, a^H_{t+m}]$.

## 3 Related work

The problem of automated skill discovery is the most commonly addressed issue within the field of HRL. As the cornerstone, the *Options framework* [24] laid a base for many subsequent approaches to build on. The research was originally focused mostly on approaches for discrete state-space environments [2,4,7,14,16,20,24], as being significantly easier to tackle. Only the recent approaches, supported by the improvements in deep neural networks, were able to solve continuous domains [6,10,12,13,17,19], as we also do in our work.

The process of acquiring skills differs widely among individual works, and is often supported by a surrogate reward signal. In the simplest case, older methods [20,24] specify and train the skill behaviors manually as a series of RL agents. The frequency-based [7,14] and graph-based [16,17] approaches try to identify subgoals, and then use a simple state-based reward to train a skill for

reaching them. Others employ more sophisticated rewards in order to train better specialised skills [6,13]. Analogously, we also construct a specialised reward signal for each missing skill, so that each new skill fills in the specific gap in the skill set.

The vast majority of research operates strictly on two leveled hierarchies [1,2,6,10,11,13,14,15,16,17,18,24], although there have been successful demonstrations of learning a multi-leveled hierarchy [4,12]. Our ASA framework was also designed for two levels. However, contrarily to [2,6,18], there is no fundamental constraint preventing it from being deployed on multi-level hierarchies.

In terms of a new skill-training method, our approach is similar to the *Skill-chaining* algorithm [10]. Both approaches train a skill to reach from given starting states to end states. However, their skill training is processed strictly in the pre-training phase, and produces only a linearly aligned stream of skills. Should their algorithm be deployed on a highly non-linear environment, it would most probably fail to find the suitable 'skill-chain'. On the other hand, ASA can produce versatile skills, and furthermore it can do so even after the pre-training phase is finished.

A parallel to our work can be seen in the algorithm [23], which also enables to enrich the hierarchy by new skills. However, their approach is strictly dependant on the hand-crafted curriculum of tasks supported by pre-defined stochastic grammar from which the new skills are generated, which effectively limits the capabilities of new tasks. The key advantage of ASA compared to this approach is the ability to train even behaviors that have not been considered in advance, as it is not bounded by the curriculum.

One of the latest contributions, which we consider to be the state-of-the-art for the task we are solving, is the HiPPO algorithm [13]. It also trains skills during the high-level training, as we do. It uses an approximate hierarchical policy gradient to directly train both skills *and* a high-level agent from the scratch at the same time. The only engineering choice is the latent dimension, which effectively regulates the number of skills to be trained. In their work the authors show that HiPPO consistently surpasses other similarly focused algorithms.

## 4 Our approach

In this paper we present the improvements of our method *Adaptive Skill Acquisition* (ASA) [8]. The key function of ASA is to detect the inefficiencies within a current skill set, and address them by adding a new skill *in the midst* of training of the high-level agent. ASA is designed not as a closed, self-contained architecture, but rather as a component that can be plugged into almost any existing HRL architecture, or those yet to come.

As discussed in the previous section, the majority of current HRL architectures employ a pre-training phase, after which the skills are fixed – which can hurt the system if the skill set is not optimal. Some useful skills, especially in spatial tasks, can be identified only after the high-level agent explores a sufficient part of the state space, e.g. discovering a new section of a map which the original skills did not reflect upon. By generating the skills with ASA even after the pre-

training phase, the system can acquire new skills according to the real current needs of the high-level agent.

The process of ASA is composed of three key conceptual steps:
**1) Identification of a missing skill** is performed by self-observation of the high-level agent. During the training we try to detect sequences of high-level actions which occur significantly more often than expected. Such behavior of the agent hints at a regularity in the MDP, for which no skill was trained. Hence the regularity is only modelled by reoccurring sequence of high-level actions, which can result in a highly suboptimal solution. A frequently executed sequence of skills hence serves as a candidate for a new skill.
**2) Training of the new skill** consists of standard RL training to solve an MDP that represents the new skill. However, this MDP has to be constructed dynamically and automatically, so that ASA can operate autonomously. The most important aspect of this step is constructing a robust reward function that will lead the new skill-agent towards the desired behavior.
**3) Integration of the new skill** into the overall HRL architecture will finally allow the high-level agent to use the new skill. If the agent is modelled using a neural network, this essentially means adding a new output unit to the partially trained network. Further specification of a termination criterion for the new skill will then allow it to be used more efficiently.

The described processes represent the core of Adaptive Skill Acquisition – please refer to our previous paper [8] for comprehensive description of these components. In the following sections we cover the improvements that were implemented in order to make ASA more robust, efficient, and reusable.

### 4.1 Normalization of sequence frequency

In the skill-identification phase we track the sequences of high-level actions. For each such sequence $\delta = [a_t^H, \dots, a_{t+m-1}^H]$, which is of an arbitrary length $m$, we gather the number of times it occurred $C(\delta)$. The most frequently used sequence, which would serve as a candidate for new skill, could be naively picked as the one with the highest $C(\delta)$.

However, the shorter sequences tend to naturally occur more often. The counts thus need to be normalized in order to account for this disproportion. We improve on our previous work by introducing a new null-count $C_H(\delta)$, which can now account for batch-training methods. We denote a batch by $\mathcal{T} = \{\tau_1, \tau_2, \dots\}$, being a set of agent's trajectories $\tau_j$. The improved null-count for a sequence $\delta$ of length $m$ gathered during batch $\mathcal{T}$ is computed as:

$$C_H(\delta) = \left( \sum_{\tau \in \mathcal{T}} |\tau| \ - \ (m-1)|\mathcal{T}| \right) \prod_{i=1}^{m} p(\delta_i)$$

where $\delta_i$ denotes $i^{\text{th}}$ step of $\delta$, and $p(\delta_i)$ is empirical probability of invoked skills.

The quantity $C_H(\delta)$ describes how many times $\delta$ is expected to occur if a random 'null' policy was used instead of real $\pi^H$. We can thus use it to normalize the count of the sequence by computing $f(\delta) = C(\delta)/C_H(\delta)$. This $f$-score quantity no longer favors shorter sequences, and can be used to determine the best candidate for a new skill.

### 4.2 Skill-training reward definition

Moving on to the training of the new skill based on the selected sequence $\delta$, we aim it to *perform the same state transition as $\delta$ did*, but to perform it more optimally. During the skill-identification step, we collect the start-states $S_{\text{start}}(\delta) = \{s_s^{(1)}, \ldots, s_s^{(C(\delta))}\}$ in which each occurrence of $\delta$ started, as well as end-states $S_{\text{end}}(\delta) = \{s_e^{(1)}, \ldots, s_e^{(C(\delta))}\}$ in which each occurrence of $\delta$ ended up. These two sets help us to specify the MDP for the new skill.

The training of the new agent is realised by spawning the agent in a randomly picked $s_s^{(i)} \in S_{\text{start}}(\delta)$, and only rewarding it upon reaching the corresponding $s_e^{(i)}$. Of course, reaching of a specific state is not achievable in continuous state-spaces, and a generalization into an *end-region* is needed. This was originally accomplished by setting a simple distance threshold, and rewarding the agent if $\|s_t - s_e^{(i)}\|_2 < \epsilon$. However, a careful tuning of $\epsilon$ had to be done for each new environment, since the relative distances of states can widely differ.

We improve on this approach by constructing a new reward function, which is agnostic to the choice of environment. First, we employ a technique of batch normalization $\phi(s)$ [9] on the state space $S$, typically used in deep learning. It is used to normalize each dimension of the input (i.e. state) to have zero mean and unit variance. We use it to flatten the difference between relative distances within each dimension of the state space. After the normalization, we know that the distance between any two normalized states $\phi(s_x)$ and $\phi(s_y)$ averages to $\approx 1$ in each dimension.

Secondly, we define the end-region, which the agent aims to reach, as a hyper-cube centered in a normalized version of the desired end-state $\phi(s_e^{(i)})$, with a side of $2\epsilon$. This yields a formal definition of the reward function:

$$R(s_t, a_t) = \left[ \max_{d=1..\dim(S)} \left| \phi(s_t)_d - \phi(s_e^{(i)})_d \right| < \epsilon \right]_1$$

where $\dim(S)$ is the dimensionality of state space, subscript $d$ represents the $d$-th dimension of the given vector, and $[\,\cdot\,]_1$ is the indicator function. This formulation in practice means that the agent is rewarded if each dimension of the current state (after normalization) is not more than $\epsilon$ away from the desired value.

The usage of batch normalization ensures that we can set $\epsilon$ to a reasonable value, while being agnostic to the choice of the environment. E.g. setting $\epsilon = 0.1$ means that roughly 90% match between the current and the desired state in each dimension is needed for agent's success. Moreover, using this hyper-cube based reward ensures that a partial match is required in *each* dimension. This is essential if some component of a state has a greater importance, since its error cannot be ignored even if other components have a prefect match.

### 4.3 Skill-stopping criteria

Current HRL architectures usually choose to execute the skills for a fixed amount of steps [5,6,12,18,19]. Some, however, use a state-based function for skill termination [4,10,24], randomized length [13], or other criteria. In order to increase

the compatibility of ASA even further and accommodate for all aforementioned methods, we introduce the concept of *skill-stopping functions* to the framework:

$$f_i^{\text{stop}}\left(s_{t-c}, a_{t-c}, \ldots, s_t, a_t\right) \in \{true, false\}$$

The skill-stopping function accepts the whole trajectory of skill since time $t-c$, at which it started, until the current time $t$, which can be used to implement any stopping criterion from the relevant research. Furthermore, if needed, each skill $\pi_i^L$ can have separate stopping function $f_i^{\text{stop}}$.

The stopping-function of the new skill will directly follow what the skill was trained for – reaching the end-regions $S_{\text{end}}(\delta)$. This behavior is hence identical to terminating the episodes during the new skill's training phase, when the end of an episode was determined by the surrogate reward function $R(s_t, a_t)$. We thus alternate on $R$'s equation to create a stopping function for the new $(n+1)$-th skill:

$$f_{n+1}^{\text{stop}}\left(..., s_t\right) = \left(\exists s_e^{(i)} \in S_{\text{end}}(\delta) : \max_{d=1..\dim(S)} \left|\phi(s_t)_d - \phi(s_e^{(i)})_d\right| < \epsilon\right) \vee \left(c > t_{\max}\right)$$

We still included the condition $c > t_{\max}$ to limit the maximal execution time of the skill, in case it fails to reach the desired end-region.

## 5 Results

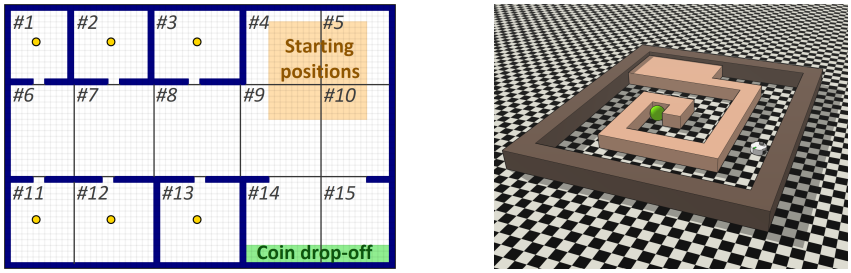
We now present new experiments and results, which significantly extended our previous pilot tests. We tested our method on two distinctive environments, and added a comparison to one of the latest HRL architectures – HiPPO [13]. Our experiments were mainly focused on two aspects:

- What is the overall ASA performance and how does it compare to HiPPO?
- What is the quality of a newly trained skill?

### 5.1 Environments and training setup

For our experiments we chose two environments which on purpose differ in numerous aspects, such as continuity, observability, skill types, etc. Our goal was to demonstrate that ASA can be used universally, not only in a single type of tasks. We thoroughly focused on sparse-reward environments.

**Coin-gatherer:** The agent in Coin-gatherer environment operates in a grid-world of size 68×46 tiles depicted in Fig. 1a. There are four possible actions (N,S,W,E). Some rooms in the map contain *coins* which have to be delivered to the *drop-off area*. The agent can only carry one coin at a time, and is rewarded only upon delivering the coin. The agent knows its position and the position of all remaining coins. The HRL architecture is realised by skills trained to reach




(a) (b)

**Fig. 1:** Environments: (a) *Coin-gatherer* map – yellow dots are coins, numbers represent 15 ideal skill regions; (b) *Maze-bot* – one of six possible mazes, the green sphere represents the goal position.

individual regions of the map, marked #1–#15 in Fig. 1a, plus four skills identical to the atomic actions. This resembles the earlier approaches [2,7,14,16,18]. Skills #14 and #15 are deliberately left out from the skill set, making a defect in the pre-trained hierarchy. Coin-gatherer environment is hence discrete, fully observable, using goal-based skills, and features a high number of 17 skills.

**Maze-bot:** The agent represents a vacuum-cleaner-like robot in a continuous environment. Its goal is to solve a given maze, using wheel torques as atomic actions. There is a total of six different mazes (one of them shown in Fig. 1b), and a maze is chosen randomly at beginning of each episode. The agent uses only a LIDAR-like sensor to detect its surroundings, with a range roughly 6 times the robot's size. However, it does *not* have an orientation sensor (compass), or the knowledge which maze it has been placed in. A reward of 1 is given if the agent reaches the maze's goal point, and $-0.05$ penalty per step otherwise. The HRL architecture is realised by locomotion skills to efficiently move a larger distance forward/backward, or turn left, but not a skill to turn right. Maze-bot environment, in contrast to Coin-gatherer, is continuous, partially observable, randomized, and uses a small number of skills which are behavioral.

In both environments, all agents are trained using TRPO algorithm [22]. The policies, as well as the policy-baseline function, are modelled using neural networks with two hidden layers. They are optimised after each training iteration – batch of 5000 high-level steps ($\approx$ 50 episodes for Coin-gatherer, $\approx$ 70 episodes for Maze-bot). The metric for the evaluation of all models was the *average discounted reward*, i.e. $G(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, and the results were averaged over 8 trials with different random seeds. The implementation of ASA was made possible by the *Garage* framework [3].

### 5.2 Overall performance

Since the primary goal of ASA is to *improve* an existing architecture, in Experiment 1 we focused on the overall gain of ASA to the pre-trained HRL agent ('Base run' without ASA). As seen in Fig. 2a and 3a, adding a new skill identified by ASA consistently and significantly increases the performance of the agent. In case of *Coin-gatherer*, this means an increase from 4.4 to 5.6 delivered coins, out of 6 possible. The *Maze-bot*'s increase by 0.81 translates to paths shorter by 16.2 high-level steps, on average. The shaded areas in these plots represent the 25–75 percentile for *Coin-gatherer*, and 5–95 percentile for *Maze-bot* (since its training was more stable).

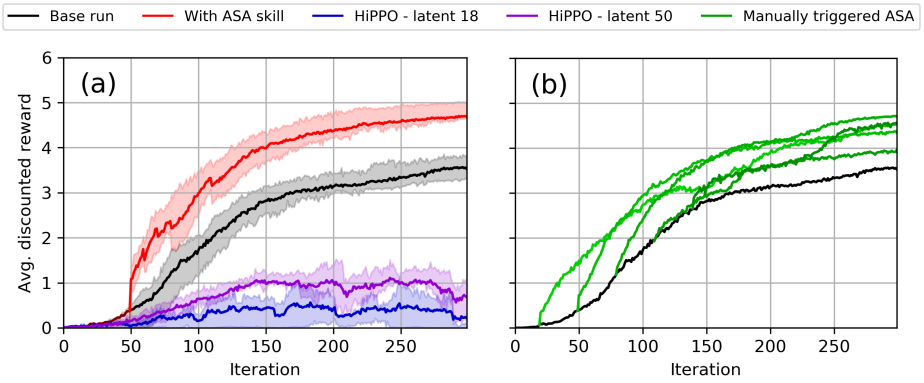



**Fig. 2:** *Coin-gatherer* environment – results after using ASA to add new skill. New skill was added (a) according to ASA computation, or (b) at other times overriding ASA's decision. Comparison with HiPPO for reference.

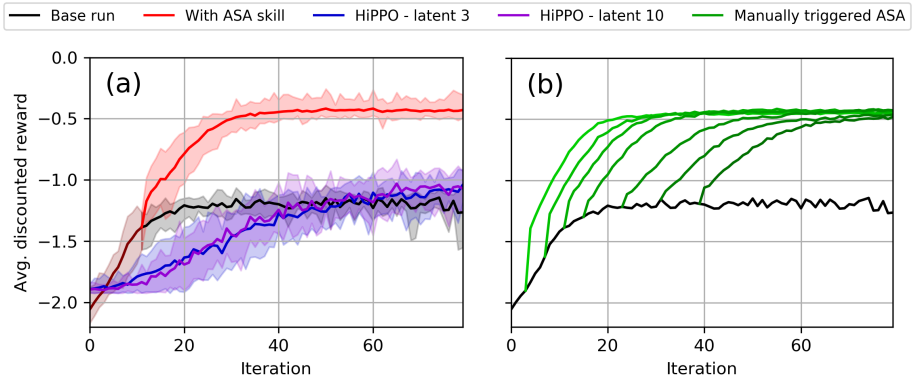



**Fig. 3:** *Maze-bot* environment – results after using ASA to add new skill. New skill was added (a) according to ASA computation, or (b) at other times overriding ASA's decision. Comparison with HiPPO for reference.

As mentioned before, the two environments use fundamentally different types of skills. Since ASA was successful in both cases, it suggests that it is able to train both *goal-based* and *behavioral* types of skills. To the best of our knowledge, there has not yet been a published model that would be shown to demonstrate such behavior.

To the best of our knowledge, there is no algorithm other than ASA that would autonomously add new skills to existing architecture, which we could use for direct comparison. Hence, we compare it with its ideologically closest relative – HiPPO [13], which autonomously creates whole architecture, and is one of few models capable of skill training even *during* high-level training, as we do in our approach. Its slower start can be explained by initially untrained skills, compared to pre-trained (imperfect) skill set in the Base run. However, it clearly did not manage to identify all useful skills, and hence scores significantly worse compared to the ASA-powered model. We also tried to increase the latent dimension of HiPPO, which effectively controls how many skills are trained, but no further improvement was observed.

The plots in Fig. 2a and 3a show the results of a full-stack approach, i.e. all key steps (decision, training, integration) were performed by ASA. On the other hand, in Fig. 2b and 3b we depict the usage of ASA, but with deactivated component that decides *when* to add the new skill. We can see that even though ASA was triggered manually, it was still able to identify and train a reasonably useful skill.

### 5.3 Quality of the new skill

Since the new skill is the key component of ASA method, in Experiment 2 we aimed to evaluate its quality. To do so, we compared it with *ideal* and *bad* skills, which serve as the upper and lower bounds for estimating the usefulness of the new skill. The *ideal* skills were manually constructed to optimally enrich the skill set: a policy for reaching regions #14 and #15 in *Coin-gatherer* environment, and a skill for turning right in *Maze-bot* environment. The *bad* skills were uniform random policies for both environments.

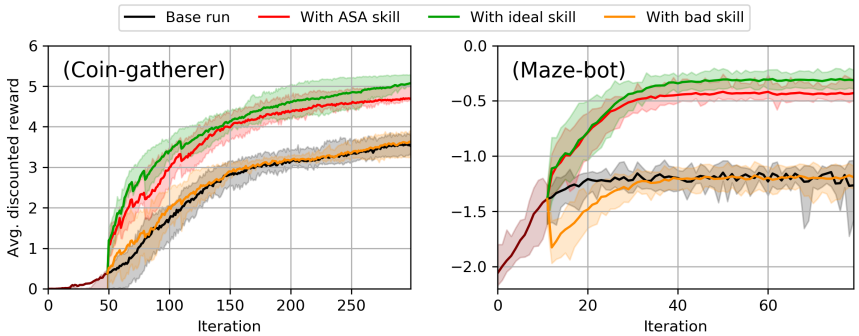



**Fig. 4:** Integration of ASA-trained skill, an ideal skill that optimally enriches the skill set, and an intentionally useless bad skill.

Fig. 4 shows the comparison between the ideal, bad, and ASA-trained skills, from which we can draw several conclusions. First and foremost, the ASA skill identification performs very well, falling short only slightly behind the ideal skills. If we consider the ideal and bad skills as a range bounding the fitness of the skill, we can express that ASA-trained skill scored solid 73.2% of the possible performance in *Coin-gatherer*, and excellent 88.7% in *Maze-bot*. Second, the unimproved performance of a bad skill proves that adding *any* skill is not sufficient for an increased success rate, and hence the success of ASA is not only coincidental. Finally, we can see that adding a useless skill does not hurt the model performance in the long-term view. Hence, even if ASA was used on top of an optimal skill set, and would add a falsely identified missing skill, it would not decrease the overall performance.

## 6 Conclusion and future work

In this paper we presented our *Adaptive Skill Acquisition* model for adding new skills to (possibly suboptimal) HRL hierarchies, and the improvements that make it more robust, efficient, and reusable. Using two distinctive environments, both discrete and continuous, we demonstrated that the new skill trained by ASA can significantly improve the performance of a HRL agent, if it started with a suboptimal skill set. The new skills identified by ASA are only slightly inferior to the optimal ones, proving high quality of the skill-identification component. We compared our method with the state-of-the-art HiPPO algorithm [13] which trains the whole skill set, and showed that ASA skill identification outperforms the older approach by a significant margin.

As a continuation of our work, we would like to adapt the ASA framework even for UMDP-based architectures, such as [12,19]. We will also consider a pseudo-rehearsal technique [21] for the skill integration, which could help speed up the adaptation of the new skill by the high-level agent.

## References

1. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: AAAI Conference on Artificial Intelligence (2017)
2. Bakker, B., Schmidhuber, J.: Hierarchical reinforcement learning with subpolicies specializing for learned subgoals. In: International Conference on Neural Networks and Computational Intelligence. pp. 125–130 (2004)
3. Garage contributors: Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage (2019)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of Artificial Intelligence Research **13**(1), 227–303 (2000)
5. Dillinger, V.: Abstract state space construction in hierarchical reinforcement learning. Ph.D. thesis, Comenius University in Bratislava (2019)
6. Florensa, C., Duan, Y., Abbeel, P.: Stochastic neural networks for hierarchical reinforcement learning. In: International Conference on Learning Representations (2017)

7. Goel, S., Huber, M.: Subgoal discovery for hierarchical reinforcement learning using learned policies. In: Florida AI Research Society Conference. pp. 346–350 (2003)
8. Holas, J., Farkaš, I.: Adaptive skill acquisition in hierarchical reinforcement learning. In: International Conference on Artificial Neural Networks. pp. 383–394. Springer (2020)
9. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)
10. Konidaris, G., Barto, A.G.: Skill discovery in continuous reinforcement learning domains using skill chaining. In: Advances in Neural Information Processing Systems. pp. 1015–1023 (2009)
11. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: Advances in Neural Information Processing Systems. pp. 3675–3683 (2016)
12. Levy, A., Konidaris, G., Platt, R., Saenko, K.: Learning multi-level hierarchies with hindsight. In: International Conference on Learning Representations (2018)
13. Li, A.C., Florensa, C., Clavera, I., Abbeel, P.: Sub-policy adaptation for hierarchical reinforcement learning. In: International Conference on Learning Representations (2020)
14. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: International Conference on Machine Learning. vol. 1, pp. 361–368 (2001)
15. McGovern, E.A., Barto, A.G.: Autonomous discovery of temporal abstractions from interaction with an environment. Ph.D. thesis, University of Massachusetts at Amherst (2002)
16. Menache, I., Mannor, S., Shimkin, N.: Q-cut—dynamic discovery of sub-goals in reinforcement learning. In: European Conference on Machine Learning. pp. 295–306 (2002)
17. Metzen, J.H., Kirchner, F.: Incremental learning of skill collections based on intrinsic motivation. Frontiers in neurorobotics **7**, 11 (2013)
18. Moerman, W.: Hierarchical reinforcement learning: Assignment of behaviours to subpolicies by self-organization. Ph.D. thesis, Cognitive Artificial Intelligence, Utrecht University (2009)
19. Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3303–3313 (2018)
20. Parr, R., Russell, S.J.: Reinforcement learning with hierarchies of machines. In: Advances in Neural Information Processing Systems. pp. 1043–1049 (1998)
21. Robins, A.: Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science **7**(2), 123–146 (1995)
22. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897 (2015)
23. Shu, T., Xiong, C., Socher, R.: Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. In: International Conference on Learning Representations (2018)
24. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112**, 181–211 (1999)