Budgeted Hierarchical Reinforcement Learning

Aurélia Léon and Ludovic Denoyer
Sorbonne Universités,
UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

***Abstract*—In hierarchical reinforcement learning, the framework of options models sub-policies over a set of primitive actions. In this paper, we address the problem of discovering and learning options from scratch. Inspired by recent works in cognitive science, our approach is based on a new budgeted learning approach in which options naturally arise as a way to minimize the *cognitive effort* of the agent. In our case, this effort corresponds to the amount of information acquired by the agent at each time step. We propose the *Budgeted Hierarchical Neural Network* model (BHNN), a hierarchical recurrent neural network architecture that learns latent options as continuous vectors. With respect to existing approaches, BHNN does not need to explicitly predefine sub-goals nor to *a priori* define the number of possible options. We evaluate this model on different classical RL problems showing the quality of the resulting learned policy.**

I. Introduction

Reinforcement Learning (RL) is one of the key problems in machine learning, and the interest of the research community has been recently renewed with the apparition of models mixing classical reinforcement learning techniques and deep neural networks. These new methods include for example the DQN algorithm [1], the use of recurrent architectures with policy gradient models [2], or actor-critic algorithms [3].

Works in cognitive science have long emphasized that human or animal behavior can be seen as a hierarchical process, in which solving a task amounts to sequentially solving sub-tasks [4]. Examples of those in a simple maze environment are *go to the door* or *go to the end of the corridor*; these sub-tasks themselves correspond to sequences of primitive actions or other sub-tasks.

In the computer science domain, these works have led to the *hierarchical reinforcement learning* paradigm [5]–[7], motivated by the idea that it makes the discovery of complex skills easier. One line of approaches consists in modeling sub-tasks through *options* [8], giving rise to two main research questions: (a) choosing the best suited option and then (b) selecting actions to apply in the environment based on the chosen option. These two challenges are respectively mediated using a high-level and a low-level controllers.

In the Reinforcement Learning (RL) literature, different models have been proposed to solve these questions. But the additional question of *discovering options* is not solved yet: the hierarchical structure in the majority of existing models has to be manually constrained, e.g. by predefining possible sub-goals [9]. Learning automatic task decomposition, without supervision, remains an open challenge in the field of RL. The difficulty of discovering hierarchical policies is emphasized by the fact that it is not well understood *how* they emerge in behaviors.

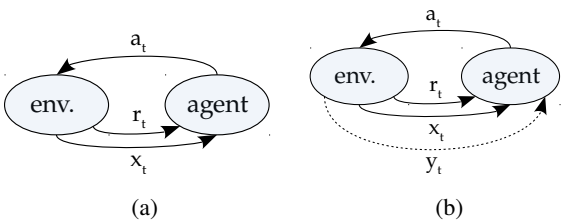


Figure 1: (a) A typical RL setting where the agent receives a reward r_t and an observation x_t from the environment at each time step, then executes an action a_t. (b) The setting used for the BHNN model, where the agent can ask for an additional information y_t.

Recent studies in neurosciences suggest that a hierarchy can emerge from *habits* [10]. Indeed, [11] and [12] distinguish between *goal-directed* and *habitual* action control; the latter not using any information from the environment. The underlying idea is that the agent first uses goal-directed control, and then progressively switches to habitual action control since goal-directed control requires a higher *cognitive cost* to process the acquired information[1].

Based on this idea, we propose the BHNN architecture (Budgeted Hierarchical Neural Network). In this framework, the agent has access to different amounts of information coming from the environment. More precisely, we consider that at each time step t, the agent can use a basic observation denoted x_t used by the *low-level* controller as in classical RL problems. In addition, it can also choose to acquire a supplementary observation y_t, as illustrated in Fig. 1. This extra observation will provide more information about the current state of the system and will be used by the *high-level* controller, but at a higher "cognitive" cost. On top of this setting, we assume that options will naturally emerge as **a way to reduce the overall cognitive effort generated by a policy**. In other words, by constraining the amount of high-level information used by our system (i.e the y_t), BHNN will learn a hierarchical structure where the resulting policy is a sequence of low-cost sub-policies. In that setting, we

[1] In that case, the cognitive cost is the time the animal spends to decide which action to take.

show that the structure of the resulting hierarchical policy can be seen as a sequence of intrinsic options [13], i.e. vectors in latent space.

The contributions of the paper are threefold:

- We propose a new assumption about hierarchy discovery, arguing that it is a consequence of learning a trade-off between policy efficiency and cognitive effort. The hierarchical structure emerges as a way to reduce the overall cost.
- We define a new model called BHNN that implements this idea as a hierarchical recurrent neural network for which, at each time step, two observations are available at two different prices. This model is learned using reinforcement learning algorithms over a budgeted objective.
- We propose different sets of experiments in multiple settings showing the interests of the model.

The paper is organized as follows: related works are presented in Section II. We define the background for RL and for recurrent RL methods in Section III. We introduce the BHNN model and the learning algorithm for the budgeted problem in Section IV. Experiments are presented and discussed in Section V. At last, Section 6 concludes and opens perspectives.

II. Related Work

The closest architecture to BHNN is the Hierarchical Multiscale Recurrent Neural Network [14] that discovers hierarchical structures in sequences. It uses a binary boundary detector learned with a straight-through estimator, similar to the acquisition model (see Section IV-A) of BHNN.

More generally, **Hierarchical Reinforcement Learning** [5]–[7] has been the surge of many different works during the last decade since it is deemed as one solution to solve long-range planning tasks and to allow the transfer of knowledge between tasks. Many different models assume that subtasks are *a priori* known, e.g., the MAXQ method in [6]. The concept of **option** is introduced by Sutton et al. in [8]. In this architecture, each option consists of an initiation set, its own policy (over primitive actions or other options), and a termination function which defines the probability of ending the option given a certain state. The concept of options is at the core of many recent articles. For example, the authors of [9] propose an extension of the Deep Q-Learning framework to integrate hierarchical value functions using intrinsic motivation to learn the option policies. But in these different models, the options have to be manually chosen *a priori* and are not discovered during the learning process. Still in the options framework, options (both internal policies and the policy over options) can be discovered without supervision [15], [16], using respectively the Expectation Maximization algorithm and the option-critic architecture. Our contribution differs from these last two in that the BHNN model does not have a fixed discrete number of options and rather uses an "intrinsic option" represented by a latent vector. Moreover, we clearly state how a hierarchy arise by finding a good trade-off between efficiency and cognitive effort.

Close to our work, the concept of cognitive effort was introduced in [17] (but with discrete options), while intrinsic options, i.e. options as latent vectors, where used in [13].

At last, some articles propose hierarchical policies based on **different levels of observations**. A first category of models is those that use open-loop policies i.e. do not use observation from the environment at every time step. A model that mixes open-loop and closed-loop control while considering that sensing incurs a cost is proposed in [18]. Some models focus on the problem of learning **macro-actions** [19], [20]: in that case, a given state is mapped to a sequence of actions. The simpler framework FiGAR [21] enables the agent to repeat the same action over several time-step. Another category of models divides the state space into several components. For instance, the Abstract Hidden Markov Model [22] is based on discrete options defined on each space region. In [23], a low-level controller is used that has only access to the proprioceptive information, and a high-level controller has access to all observations. [24] use a similar idea of factoring the state space into two components, and learn a stochastic neural network for the high-level controller. The *blind setting* of the BHNN model, described in Section V-B, is similar to (stochastic) macro-actions, but open-loop policies are rather limited in complex environments. The general BHNN architecture is more comparable to works using two different observations, however, those models do not learn *when* to use the high-level controller.

III. Background

A. (PO-) Markov Decision Processes and Reinforcement Learning

Let us denote a Markov Decision Process (MDP) as a set of states $\mathcal{S}$, a discrete set of possible actions $\mathcal{A}$, a transition distribution $P(s_{t+1}|s_t, a_t)$ and a reward function $r(s_t, a_t) \in \mathbb{R}^+$. We consider that each state s_t is associated with an observation $x_t \in \mathbb{R}^n$, and that x_t is a partial view of s_t (i.e POMDP), n being the size of the observation space. Moreover, we denote P_I the probability distribution over the possible initial states of the MDP.

Given a trajectory $x_0, a_0, x_1, a_1, ..., x_t$, a policy is defined by a probability distribution such that $\pi(x_0, a_0, x_1, a_1, ..., x_t, a) = P(a|x_0, a_0, x_1, a_1, ..., x_t)$ which is the probability of each possible action a at time t, knowing the history of the agent.

B. Learning with Recurrent Neural Network

Let us denote $\gamma \in]0, 1]$ the discount factor, and $R_t = \sum_{k=t}^{T-1} \gamma^{k-t} r(s_k, a_k)$ the discounted sum of rewards (or discount return) at time t, corresponding to the trajectory $(s_t, a_t, s_{t+1}, a_{t+1}, ..., s_T)$ with T the size of the

trajectories sampled by the policy[2]. Note that $R_0 = \sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ corresponds to the classical discount return.

We can define the reinforcement learning problem as the optimization problem such that the optimal policy π^* is computed by maximizing the expected discounted return $J(\pi)$:

$$J(\pi) = \mathbb{E}_{s_0 \sim P_I; a_0, \ldots, a_{T-1} \sim \pi} [R_0] \quad (1)$$

where s_0 is sampled following P_I and the actions are sampled based on π.

Different learning algorithms aim at maximizing $J(\pi)$, and each one of them can be used to learn the BHNN.

a) Policy gradient: In the case of policy gradient techniques, if we consider that, for sake of simplicity, π also denotes the set of parameters of the policy, the gradient of the objective can be approximated with:

$$\nabla_\pi J(\pi) \approx \frac{1}{M} \sum_{m=1}^{M} \sum_{t=0}^{T-1} \nabla_\pi \log \pi(a_t | x_0, a_0, ..., x_t) \left(R_t - b_t\right) \quad (2)$$

where M is the number of sampled trajectories used for approximating the gradient using Monte Carlo sampling techniques, b_t is a variance reduction term at time t estimated during learning, and we consider that future actions do not depend on past rewards (see [2] for details on recurrent policy gradients).

b) Asynchronous advantage actor critic: The asynchronous advantage actor critic (A3C) [25] algorithm makes an estimation of the value function $V(s_t)$. An update of the parameters of both the policy and the value function is done every n-step. At time t, the update of the policy can be seen as $\nabla_\pi \log \pi(a_t | x_0, a_0, ..., x_t) A(x_t, a_t)$ where A is an estimate of the advantage function given by $\sum_{i=0}^{k-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^k V(x_{t+k}) - V(x_t)$ where $k \leq n$. To stabilize learning, several copies of the actor-learners are executed asynchronously and the parameters updates are sent to a central model.

IV. Budgeted Hierarchical Neural Network

A. The BHNN Architecture

In a typical (PO-)MDP setting, the agent uses an observation x_t from the environment at every time step. In contrast, in the BHNN architecture, the agent always uses a *low-level* observation x_t, but can also choose to acquire a *high-level* observation y_t that will provide a more relevant information, as illustrated in Figure 1. This situation corresponds to many practical cases: for example a robot that acquires information through its camera (x_t) can sometimes decide to make a complete scan of the room (y_t); a user driving a car (using x_t) can decide to consult its map or GPS (y_t); a virtual agent taking decisions in a virtual world (based on x_t) can ask instructions from a human (y_t), etc. Note that a particular case (called *blind setting*) is when x_t is empty, and y_t is the classical observation over the environment. In that case, the agent will decide whether it wants to use the current observation, as in the *goal directed* vs *habits* paradigm [11], [12].

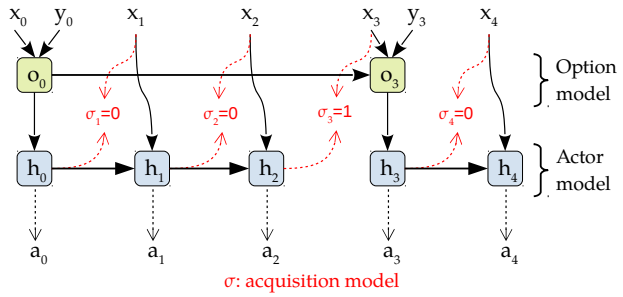


Figure 2: The BHNN Architecture. Arrows correspond to dependencies, dashed arrows correspond to sampled values. Note that in this example, $\sigma_3 = 1$ so the model decides to acquire y_3 and computes a new option o_3. When $\sigma_t = 0$ (for $t \in \{1, 2, 4\}$ in this example) the model does not use y_t and keeps the same option.

The structure of BHNN is close to a hierarchical recurrent neural network with two hidden states, o_t and h_t, and is composed of three components. The **acquisition model** aims at choosing whether observation y_t has to be acquired or not. If the agent decides to acquire y_t, the **option model** uses both observations x_t and y_t to compute a new option denoted o_t as a vector in an option latent space. The **actor model** updates the state of the actor h_t and aims at choosing which action a_t to perform.

We now formally describe these three components. A schema of the BHNN architecture is provided in Fig. 2, and the complete inference procedure is given in Algorithm 1. Note that for sake of simplicity, we consider that the last chosen action a_{t-1} is included in the low-level observation x_t, as often done in reinforcement learning, and we avoid to explicitly write the last chosen action in all equations. Relevant representations of x_t and y_t are learned through neural network – linear models in our case. In the following, the notations x_t and y_t directly denote these representations, used as inputs in the BHNN architecture.

a) Acquisition Model: The acquisition model aims at deciding whether a new high-level observation y_t needs to be acquired. It draws $\sigma_t \in \{0, 1\}$ according to a Bernoulli distribution with $P(\sigma_t = 1) = sigmoid(f_{acq}(h_{t-1}, x_t))$. If $\sigma_t = 1$, the agent will use y_t to compute a new option (see next paragraph), otherwise it will only use x_t to decide which action to apply[3].

[2]We describe finite-horizon problems where T is the size of the horizon and $\gamma \leq 1$, but the approach can also be applied to infinite horizon problems with discount factor $\gamma < 1$.

[3]In our experiments, f_{acq} is a simple linear model following the concatenation of h_{t-1} and x_t.

Algorithm 1 The pseudo code for the BHNN model.

```
procedure INFERENCE(s_0)    ▷ s_0 is the initial state
    initialize o_{-1} and h_{-1}
    for t = 0 to T do
        Acquire x_t
        acquisition model: Draw σ_t ∈ {0, 1}
        if σ_t == 1 then
            option level: Acquire y_t and update the option state o_t
            actor level: Initialize the actor state h_t
        else
            actor level: Update the actor state h_t
        end if
        actor level: Choose the action a_t w.r.t h_t
        Execute the chosen action
    end for
end procedure
```

b) Option Model: If $\sigma_t = 1$, the option model computes a new option state following $o_t = gru_{opt}(x_t, y_t, o_{last})$ in which o_{last} is the lastly computed option before time step t. gru_{opt} represents a GRU cell [26].

c) Actor Model: The actor model updates the actor state h_t and computes the next action a_t. The update of h_t depends on σ and the availability of the high-level observation y_t:

- if $\sigma_t = 0$: $h_t = gru_{act}(x_t, h_{t-1})$
- if $\sigma_t = 1$: $h_t = o_t$

The next action a_t is then drawn from the distribution $softmax(f_{act}(h_t))$[4].

Options and BHNN: We consider that every time the model chooses to acquire a high-level observation y_t, it computes a new *intrinsic option* [13], thus the name "option model" for the second component of BHNN. An intrinsic option is encoded through the continuous vector o_t each time that $\sigma_t = 1$. Then, the policy will behave as a classical recurrent policy, using the last computed o_t until a new high-level observation is acquired. In other words, when acquiring a new high-level observation, a new sub-policy is chosen depending on x_t and y_t (and eventually the previous option). The option state o_t can be seen as a latent vector representing the *option* chosen at time t, while σ_t represents what is usually called the *termination function* in option settings. In BHNN, since the option is chosen directly according to the state of the environment (more precisely on the observations of the agent x_t and y_t), there is no need to have an explicit initiation set defining the states where an option can begin.

B. Budgeted Learning for Hierarchy Discovery

Inspired by cognitive sciences [27], BHNN considers that discovering a hierarchy aims at reducing the *cognitive effort* of the agent. In our case, the cognitive effort is measured by the amount of high-level observations y_t acquired and computed by the model to solve the task, and thus by the amount of options vectors o_t computed by the model over a complete episode. By constraining our model to find a good trade-off between policy efficiency and the number of high-level observations acquired, BHNN discovers when this extra information is essential and has to be acquired, and thus when to start new sub-policies.

Let us denote $C = \sum_{t=0}^{T-1} \sigma_t$ the acquisition cost for a particular episode. We propose to integrate the acquisition cost C (or cognitive effort) in the learning objective, relying on the *budgeted learning* paradigm already explored in different RL-based applications [28], [29]. We define an augmented immediate reward r^* that includes the generated cost:

$$r^*(s_t, a_t, \sigma_t) = r(s_t, a_t) - \lambda\sigma_t \tag{3}$$

where λ controls the trade-off between the policy efficiency and the cognitive charge. The associated discounted return is denoted R_t^*, so that R_0^* will be used as the new objective to maximize. This new objective can be used in any reinforcement learning algorithm, noticing that the learning of BHNN entail the learning of both the acquisition model (i.e. a policy that choose to acquire y_t or not) and the option and actor models (i.e. the choice of action).

In case of the policy gradient algorithm, the new gradient given a trajectory $x_0, a_0, ...x_T$ is:

$$\sum_{t=0}^{T-1} (\nabla_\pi \log P(a_t|h_t) + \nabla_\pi \log P(\sigma_t|h_{t-1}, x_t)) \, (R_t^* - b_t^*) \tag{4}$$

where h_t is the recurrent state encoding the past trajectory $x_0, a_0, ..., x_t$ as defined in IV-A0c.

Note that this rule now updates both the probabilities of the chosen actions a_t and the probabilities of the σ_t that can be seen as *internal actions* and that decide if the high-level observation y_t has to be acquired or not. b_t^* is the new resulting variance reduction term as defined in [2].

The case of A3C is similar and the resulting update at time t is:

$$\nabla_\pi \log P(a_t|h_t) A^*(h_t) + \nabla_\pi \log P(\sigma_t|h_{t-1}, x_t) A^*(h_{t-1}, a_t) \tag{5}$$

where A^* is an estimate of the new advantage function associated with the augmented reward r^*

V. Experiments

A. Experimental setting

Different environments were used to evaluate BHNN. We used recurrent policy gradient to optimize the policy in most environments, except for the Atari games where

[4] gru_{act} represents a GRU cell while f_{act} is in our experiments a simple perceptron.

A3C was used (as frequently done in literature). For all experiments, we used the ADAM optimizer [30] with gradient clipping[5]. The learning rates were optimized by grid-search.

Unless precised otherwise, observations x_t and y_t are represented through linear models with a hidden layer of sizes N_x and N_y respectively, followed by an activation function $relu$, and the GRU cells are of size N_{gru} (dependent on the environment) [26].

To ensure that the policy discovers a good trade-off between accuracy and the frequency of high-level observation acquisition (and thus the number of options computed), the cost of observation y_t is set to slowly increase over time from 0 to λ. Indeed, in preliminaries experiments with a beginning cost λ too high, the model would first learn to never observe y_t and would have difficulties to improve results afterwards.

We have performed experiments in different settings that are described below. The organization of the next sections is the following:

- We first experiment our model in the *blind setting* (Section V-B) to demonstrate that BHNN is able to learn policies that can act without observing the environment at every time-step.
- We then propose a setting (Section V-C) where the agent can access both low-level and high-level observations in stochastic environments, showing how this approach is able to better improve the trade-off previously described.
- Finaly (Section V-E), we consider a particular application where the high-level observations correspond to *instructions* computed by an oracle.

B. Blind Setting

Given a POMDP (or MDP), the easiest way to design the two observations x_t and y_t needed for the BHNN model is to consider that x_t is the empty observation and y_t is the usual observation coming from the environment. This case is similar to the "*goal-directed* vs *habit action control*" paradigm and corresponds to a case in which the agent chooses either to acquire or not the observation. It also corresponds to a (stochastic) macro-actions framework: the agent chooses a sequence of actions for each observation.

Several environments were used to evaluate BHNN in this setting:

- **CartPole:** This is the classical cart-pole environment as implemented in the OpenAI Gym platform [31][6] in which observations are $(position, angle, speed, angular speed)$, and possible actions are $right$ or $left$. The reward is $+1$ for every time step without failure. In this environment, the sizes of the networks used are $N_y = 5$ and $N_{gru} = 5$.
- **LunarLander:** This environment corresponds to the Lunar Lander environment proposed in OpenAI Gym where observations describe the position, velocity, angle of the agent and whether it is in contact with the ground or not, and possible actions are *do nothing, fire left engine, fire main engine, fire right engine*. The reward is +100 if landing, +10 for each leg on the ground, -100 if crashing and -0.3 each time the main engine is fired. Here $N_y = 10$ and $N_{gru} = 10$.
- $k \times k$**-rooms:** This environment corresponds to a maze composed of $k \times k$ rooms with doors between them (see Figure 4a). The agent always starts at the upper-left corner, while the goal position is chosen randomly at each episode: it can be in any room, in any position and its position changes at each episode. The reward function is -1 when moving and +20 when reaching the goal, while 4 different actions are possible: *up, down, left* and *right*. The observation describes the agent position, the position of the doors in the room, and the goal position if the goal is in the same room than the agent (i.e the agent only observes the current room).
  Note that this environment is much more difficult than other 4-rooms problems (introduced by [8]). In the latter, there is only one or two possible goal position(s) while in our case, the goal can be anywhere. Moreover, in our case, in a more realistic setting, the agent only observes the room it is in. Here $N_y = 20$ and $N_{gru} = 10$.
- **Pong:** This environment is the game Pong from the Atari domain. As the atari games are more complex and the observation input is an image, the architecture of BHNN had to be adapted. We used the LSTM-variant of A3C algorithm [25], and observations y_t are represented through a convolutionnal network (four convolutionnal layers, each with 32 filters of size 3×3 with stride 2), and the LSTM layers are of size 256. As usualy done for these environments, each action is repeated 4 times. Results for FiGAR come from [21].

In simple environments like CartPole and LunarLander, the option model does not need to be recurrent and o_t is just dependent on x_t and y_t.

Results: We compare BHNN to a recurrent policy gradient (R-PG) with GRU cells. Note that R-PG has access to all observations (x_t and y_t) at every time step and find the optimal policy in these environments, while BHNN learns to use y_t only from time to time.

We also compare BHNN to the framework FiGAR (Fine Grained Action Repetition) [21], which enables the agent to decide how many time it will repeat the chosen action. FiGAR only use the environment observation when it must choose which action to execute and the number of repetitions. In our experiments, FiGAR uses a similar network architecture than R-PG and BHNN and the maximum number of action repetition is 10 for CartPole and

[5]An open-source version of the model is available at https://github.com/aureliale/BHNN-model.

[6]https://gym.openai.com/

		$\epsilon = 0$		$\epsilon = 0.25$	
		R	%obs	R	%obs
Cartpole	R-PG	200	1	196.0	1
	FiGAR $\lambda = 0$	200.0	0.38	185.3	0.44
	FiGAR $\lambda = 1$	199.3	0.23	184.2	0.3131
	BHNN $\lambda = 0.5$	199.7	0.06	181.6	0.26
	BHNN $\lambda = 1$	190.3	0.05	172.2	0.20
Lunar Lander	R-PG	227.3	1	109.3	1
	FiGAR $\lambda = 0$	227.5	0.33	119.6	0.31
	FiGAR $\lambda = 0.5$	220.6	0.20	108.89	0.14
	BHNN $\lambda = 0.5$	221.2	0.16	91.6	0.07
	BHNN $\lambda = 5$	210.5	0.06	90.4	0.04
3×3-rooms	R-PG	-3.3	1	-14.9	1
	FiGAR $\lambda = 0$	-5.02	0.68	-15.4	0.98
	FiGAR $\lambda = 1$	-6.9	0.44	-22.1	0.52
	BHNN $\lambda = 1$	-7.4	0.36	-16.3	0.61
Pong	LSTM-A3C	21	0.25	-	-
	FiGAR $\lambda = 0$	20.32	0.077	-	-
	BHNN $\lambda = 0.01$	19.9	0.02	-	-

Table I: Cost/reward values for the different environments, at different cost levels λ and different stochasticity levels ϵ.

Lunar Lander and 5 for 3×3-rooms. To compare FiGAR to BHNN fairly, we also use the augmented reward r^* including the cost λ.

We illustrate the quality of BHNN in Table I. There is two versions of each environment: a deterministic one ($\epsilon = 0$) and a stochastic one ($\epsilon = 0.25$) in which the movement of the agent can fail with probability ϵ: in that case, a random transition is applied.

In deterministic environments, we can see that the BHNN model is able to perform as well as classical baselines while acquiring the observation only a few times per episode: for example in the Cartpole environment, the agent needs to use the observation only 6% of the time. These results clearly show that in simple environments, there is no need to receive (observations) feedback permanently, and that a single planning step can generate several actions. Comparing BHNN with FiGAR, we see that BHNN can have roughly the same accuracy while observing the environment less frequently than FiGAR. This is due to the fact that FiGAR repeats the same action several times while a BHNN is able to discover macro-actions that mix several different actions, and thus has a higher expressivity power.

In stochastic environments, observations are used more often but even then the performances degrade much more. Indeed, due to the stochasticity, it is not possible for the agent to deduce its position knowing only a past observation and the actions choosen since then (the agent will not know anymore in which state the environment is), and the observation thus needs to be acquired more often. It demonstrates the limits of open-loop policies in non predictable environments, justifying the use of a basic observation x_t in the following.

C. Using low-level and high-level observations

As seen above, the problem of open-loop control – i.e control without any feedback from the environment – is that the stochasticity of environment cannot be "anticipated" by the agent without receiving feedback. We study

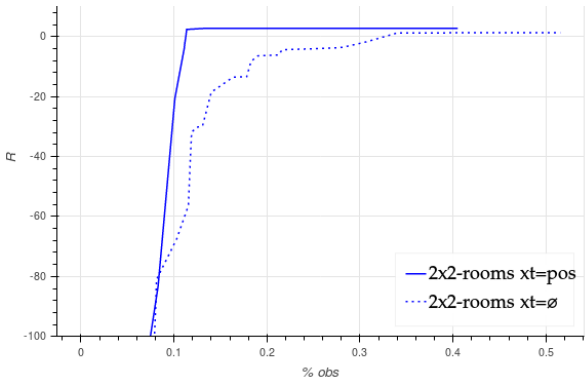


Figure 3: Reward w.r.t. cost curves for 2×2-rooms with a stochasticity level of $\epsilon = 0.25$.

here a setting in which x_t provides a simple and light information, while y_t provides a richer information. The motivation is that x_t can help to decide which action to take, without as much cognitive effort as when using the complete observation.

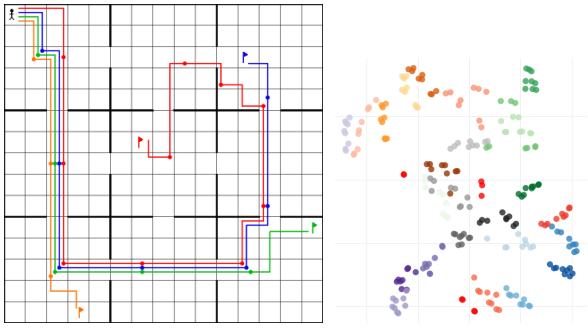
For that, we use another version of the $k \times k$-rooms environment where x_t contains the agent position in the current room, while y_t corresponds to the remaining information (i.e positions of the doors and position of the goal if it is in the room). The difference with the previous setting is that the agent has a permanent access to its position. In this version, we used $N_x = 10$, $N_y = 10$ and $N_{gru} = 10$.

In a stochastic environment ($\epsilon = 0.25$), BHNN only uses y_t 16% of the time (versus 60% in the blind setting) and even so achieve a cumulative reward of -18 (roughly the same). We can see in Fig. 3 the rewards w.r.t. cost curves obtained by computing the Pareto front over BHNN models with different percentage of observations (obtained by slowly increasing the cost level λ). We note that the drop of performance in 2×2-rooms with $x_t = position$ happens at a lower cost than the one in 2×2-rooms environment with blind setting. Indeed, with $x_t = position$, the agent knows where it is at each time step and is more able to "compensate" the stochasticity of the environment than in the first case, in which the position is only available through y_t. Like in the deterministic blind setting, the agent is able to discover meaningful options and to acquire the relevant information only once by room – see Section V-D.

This experiment shows the usefulness of permanently using a basic observation with a low cognitive cost, in contrast to temporary "pure" open-loop policies where no observation is used.

D. Analysis of option discovered

Figures 4a illustrates trajectories generated by the agent in the 3×3-rooms environment, and the positions where the options are generated. We can see that the agent learns to observe y_t only once in each room and that the agent uses the resulting option until it reaches another room.



(a) (b)

Figure 4: (a) 3×3-rooms: example of 4 trajectories (each one in different color) generated by the agent. Each point corresponds to a position where the agent decides to acquire y_t and generates a new option: the agent uses only one option per room. (b) The options latent vectors visualized through the t-SNE algorithm. Similar colors mean the goal is in similar areas in the room, except for the red points that corresponds to the options used to reach one of the four possible doors, or when the goal is just near a door.

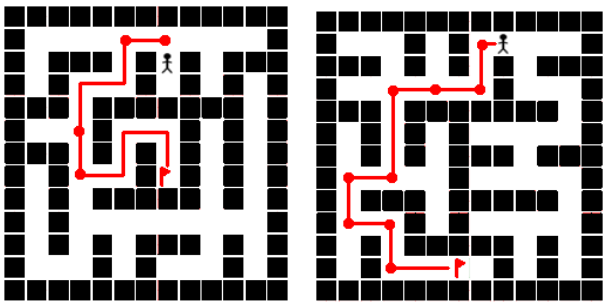
Thus the agent deducts from y_t if he must move to another room, or reach the goal if it is in the current room. Note that the agent does not go directly to the goal room because it has no information about it. Seeing only the current room, it learns to explore the maze in a "particular" order until reaching the goal room.

We have also visualized the options latent vectors using the t-SNE algorithm (Figure 4b). Similar colors (for example all green points) mean that the options computed correspond to observations for which the goals are in similar areas. We can for example see that all green options are close, showing that the latent option space effectively captures relevant information about options similarity.

E. Instructionnal Use-Case

We consider at last a setting in which y_t is an information provided by an oracle, while x_t is a classical observation. The underlying idea is that the agent can choose an action based only on its observation, or use information from an optimal model with a higher cost. To study this case, we consider a maze environment where the maze is randomly generated for each episode (so the agent cannot memorize the exact map) and the goal and agent initial positions are also randomly chosen. The observation x_t is the 9 cases surrounding the agent. The observation y_t is a one hot vector of the action computed by a simple path planning algorithm that has access to the whole map. The parameters of the model are $N_x = 10$ and $N_{gru} = 5$ (no representation is used for y_t). Note that the computation of y_t can be expensive, leading to the idea that it has a higher cognitive cost.

Two examples of generated trajectories are illustrated in Fig. 5. Figure 5b illustrates a generated trajectory **when learning is not finished**. It shows that, at a certain point, the low-controller has learned to follow a straight path but needs to ask for instructions at each cross-road, or when a change of direction is needed. Some relevant options have already emerged in the agent behavior but are not optimal. When **learning is finished** (Fig 5a), the agent now ask for instructions only at cross-roads where the decision is essential to reach the goal. Between crossroads, the agent learns to follow the corridors, which corresponds to an intuitive and realistic behavior. Note that future works will be done using outputs of more expensive models in lieu of a high-level observation y_t, and the BHNN model seems to be an original way to study the *Model Free/Model-based* paradigm proposed in neuroscience [32].



(a) (b)

Figure 5: Examples of trajectories in the maze. Each point corresponds to a position where the agent decides to acquire y_t and generates a new option: (a) Optimal learning with y_t only used at each crossroad. (b) Incomplete learning where the agent also uses y_t every time he must change direction.

VI. Conclusion and Perspectives

We proposed a new model for hierarchical reinforcement learning in which the agent chooses when to acquire a more informative – but costly – observation at each time step. The model is learned in a budgeted learning setting in which the acquisition of the additional information, and thus the use of a new option, has a cost. The learned policy is a trade-off between the efficiency and the cognitive effort of the agent. In our setting, the options are handled through learned latent representations. Experimental results demonstrate the possibility of reducing the cognitive cost – i.e. acquiring and computing information – without a drastic drop in performances. We also show the benefit of using different levels of observation and the relevance of extracted options. This work opens different research directions. One is to study if BHNN can be applied in multi-task reinforcement learning problems (the environment $k \times k$-rooms, since the goal position is randomly chosen at each episode, can be seen as a multi-task problem). Another question would be to study problems where many different observations can be acquired by the agent at different costs - e.g, many different sensors on a robot. Finally, a promising perspective is learning how and when

to interact with another expensive model, or another and more informed agent.

ACKNOWLEGMENTS

This work has been supported within the Labex SMART supported by French state funds managed by the ANR within the Investissements d'Avenir programme under reference ANR-11-LABX-65.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of the IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[3] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, 1999, pp. 1008–1014.

[4] M. Botvinick, Y. Niv, and A. C. Barto, "Hierarchically organized behavior and its neural foundations: A reinforcement-learning perspective," *cognition*, vol. 113.3, 2009.

[5] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*. Morgan Kaufmann Publishers, 1993, pp. 271–271.

[6] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning." in *ICML*. Citeseer, 1998, pp. 118–126.

[7] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," *Advances in neural information processing systems*, pp. 1043–1049, 1998.

[8] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[9] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *arXiv preprint arXiv:1604.06057*, 2016.

[10] A. Dezfouli and B. W. Balleine, "Actions, action sequences and habits: evidence that goal-directed and habitual action control are hierarchically organized," *PLoS Comput Biol*, vol. 9, no. 12, p. e1003364, 2013.

[11] M. Keramati, A. Dezfouli, and P. Piray, "Speed/accuracy trade-off between the habitual and the goal-directed processes," *PLoS Comput Biol*, vol. 7, no. 5, p. e1002055, 2011.

[12] A. Dezfouli and B. W. Balleine, "Habits, action sequences and reinforcement learning," *European Journal of Neuroscience*, vol. 35, no. 7, pp. 1036–1051, 2012.

[13] K. Gregor, D. J. Rezende, and D. Wierstra, "Variational intrinsic control," *arXiv preprint arXiv:1611.07507*, 2016.

[14] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," *arXiv preprint arXiv:1609.01704*, 2016.

[15] C. Daniel, H. van Hoof, J. Peters, and G. Neumann, "Probabilistic inference for determining options in reinforcement learning," 2016.

[16] P.-L. Bacon and D. Precup, "The option-critic architecture," in *NIPS Deep Reinforcement Learning Workshop*, 2015.

[17] ——, "Learning with options: Just deliberate and relax," 2015.

[18] E. A. Hansen, A. G. Barto, and S. Zilberstein, "Reinforcement learning for mixed open-loop and closed-loop control," in *NIPS*, 1996, pp. 1026–1032.

[19] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. Dean, and C. Boutilier, "Hierarchical solution of markov decision processes using macro-actions," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 220–229.

[20] V. Mnih, J. Agapiou, S. Osindero, A. Graves, O. Vinyals, K. Kavukcuoglu *et al.*, "Strategic attentive writer for learning macro-actions," *arXiv preprint arXiv:1606.04695*, 2016.

[21] S. Sharma, A. S. Lakshminarayanan, and B. Ravindran, "Learning to repeat: Fine grained action repetition for deep reinforcement learning," *arXiv preprint arXiv:1702.06054*, 2017.

[22] H. H. Bui, S. Venkatesh, and G. West, "Policy recognition in the abstract hidden markov model," *Journal of Artificial Intelligence Research*, vol. 17, pp. 451–499, 2002.

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

[24] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," *ICLR*, 2016.

[25] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[26] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[27] W. Kool and M. Botvinick, "A labor/leisure tradeoff in cognitive control." *Journal of Experimental Psychology: General*, vol. 143, no. 1, p. 131, 2014.

[28] G. Contardo, L. Denoyer, and T. Artières, "Recurrent neural networks for adaptive feature acquisition," in *International Conference on Neural Information Processing*. Springer International Publishing, 2016, pp. 591–599.

[29] G. Dulac-Arnold, L. Denoyer, P. Preux, and P. Gallinari, "Sequential approaches for learning datum-wise sparse representations," *Machine Learning*, vol. 89, no. 1-2, pp. 87–122, 2012.

[30] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[32] J. Gläscher, N. Daw, P. Dayan, and J. P. O'Doherty, "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning," *Neuron*, vol. 66, no. 4, pp. 585–595, 2010.