OPTIONS DISCOVERY WITH BUDGETED REINFORCEMENT LEARNING

Aurélia Léon
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
aurelia.leon@lip6.fr

Ludovic Denoyer
Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
ludovic.denoyer@lip6.fr

ABSTRACT

We consider the problem of learning hierarchical policies for Reinforcement Learning able to discover options, an option corresponding to a sub-policy over a set of primitive actions. Different models have been proposed during the last decade that usually rely on a predefined set of options. We specifically address the problem of automatically discovering options in decision processes. We describe a new learning model called *Budgeted Option Neural Network* (BONN) [1] able to discover options based on a budgeted learning objective. The BONN model is evaluated on different classical RL problems, demonstrating both quantitative and qualitative interesting results.

1 INTRODUCTION

Reinforcement Learning (RL) is one of the key problem 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 (Mnih et al., 2015) and its variants (Van Hasselt et al., 2015), the use of recurrent architectures with policy gradient models (Wierstra et al., 2010), or even approaches like Guided Policy Search (Levine & Koltun, 2013) or actor-critic algorithms (Konda & Tsitsiklis, 1999).

Research in cognitive science based on the study of human or animal behavior have long emphasized that the internal policy of such agents can be seen as a hierarchical process where solving a task is obtained by sequentially solving sub-tasks, each sub-task being treated by choosing a sequence of primitive actions (Botvinick et al., 2009). In the computer science domain, these researches have been echoed during the last decade with the apparition of the *hierarchical reinforcement learning* paradigm (Dayan & Hinton, 1993; Dietterich, 1998; Parr & Russell, 1998) and its generalization to *options* (Sutton et al., 1999). The underlying idea is to define a policy at two different levels: a level which goal is to choose between options, and a level which will select the actions to apply to the environment based on the current option. Informally, in a maze an option can correspond to an order like *go to the door*, while the actions are primitive moves (up, down, left, right). In the literature, the catalog of available options is usually specified manually which is not satisfactory.

We propose a new architecture called BONN (*Budgeted Options Neural Network*) able to simultaneously discover options and to learn how and when to use them. It is based on the idea that a good policy is a trade-off between policy efficiency and *cognitive effort*: a system will learn relevant options if these options allow it to reduce the *cognitive effort* for solving the task, without decreasing the quality of the solution. This idea is implemented here through a budgeted learning problem that encourages the BONN model to learn to acquire as few information as possible.

[1]code available here: https://github.com/aureliale/BONN-model

The contributions of the paper are: (i) We propose the BONN model able to discover options based on a Budgeted Reinforcement Learning problem where information acquisition has a cost, each option being a continuous vector in an learned latent option space. (ii) We propose a discrete variant of BONN (D-BONN) where a discrete set of options is learned, each option corresponding to a particular embedding in the latent option space. (iii) The model is tested on different RL tasks and exhibits interesting properties and a strong ability to capture relevant options.

The paper is organized as follows: we present the background in RL and on recurrent policy gradients methods in Section 2. The BONN model is presented in Sections 3.1 while the budgeted learning problem is described in Section 3.2. The variant of BONN able to extract a discrete set of options is given in Section 3.3. At last, experiments are proposed in Section 4 while the related works are presented in Section 5.

2 BACKGROUND

2.1 (PO-) MARKOV DECISION PROCESSES AND REINFORCEMENT LEARNING

Let us denote a 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 $r(s,a) \in \mathbb{R}^+$. 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 current trajectory $x_1, a_1, x_2, a_2, ..., x_t$, a policy is defined by a probability distribution such that $\pi(x_1, a_1, x_2, a_2, ..., x_t, a) = P(a|x_1, a_1, x_2, a_2, ..., x_t)$ which is the probability of each possible action a at time t, knowing the history of the agent.

2.2 LEARNING WITH RECURRENT POLICY GRADIENT

Let us denote $\gamma \in]0,1]$ the discount factor, and $R_t = \sum\limits_{k=t}^{T-1} \gamma^{k-t} r(s_k, a_k)$ the discounted sum of rewards (or discount return) at time t [2], corresponding to the trajectory $(s_0, a_0, s_1, a_1, ..., s_T)$ with T the size of the trajectories sampled by the policy [3].

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 $J(\pi)$:

$$J(\pi) = \mathbb{E}_{s_0 \approx P_I, a_0, ..., a_{T-1} \approx \pi} [R_0] \tag{1}$$

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

Different learning algorithms aim at maximizing $J(\pi)$. 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:

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

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 Wierstra et al. (2010) for details on recurrent policy gradients).

[2]Note that $R_0 = \sum\limits_{k=0}^{T-1} \gamma^k r(s_k, a_k)$ correspond to the classical discount return

[3]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 $\gamma < 1$

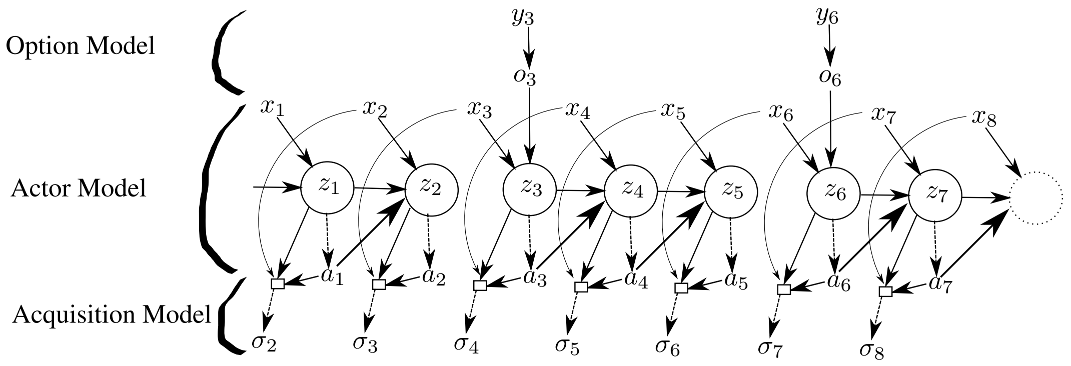

Option Model
Actor Model
Acquisition Model
y_3
o_3
y_6
o_6
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8
z_1 z_2 z_3 z_4 z_5 z_6 z_7
a_1 a_2 a_3 a_4 a_5 a_6 a_7
σ_2 σ_3 σ_4 σ_5 σ_6 σ_7 σ_8

Figure 1: The BONN Architecture. Arrows correspond to dependencies, dashed arrows correspond to sampled values. Note that when $\sigma_t = 1$ the model observes y_t and compute a new option (in this example we have $\sigma_3 = 1$ and $\sigma_6 = 1$), and that when $\sigma_t = 0$ (everywhere else in this example) the model doesn't use y_t and keeps the same option.

3 Budgeted Option Neural Network

3.1 The BONN Architecture

We consider here a particular case of POMDP where the agent always observe x_t, but can also ask for the supplementary observation y_t that will help him to decide which action to choose. This situation corresponds to many practical cases: for example a robot that acquires information through its camera (x_t) but can sometimes decide to make a complete scan of the room (y_t); a user driving a car (using x_t) but who decides to consult its map or GPS (y_t); a virtual agent taking decisions in a virtual world (based on x_t) but that can ask instructions from a human (y_t), etc. Note that x_t or y_t can be an empty observation.

We now describe the budgeted option neural network. This model is composed of three components. The underlying idea is that the first component will use the additional observations y_t to compute which *option* to use, while the second component will use the basic observations x_t and the lastly chosen *option* to sample primitive actions; the third component being used to decide when to switch between options. A new option will thus be computed each time y_t is acquired. Let us now describe how each component works: (i) The first one (or **option model**) aims at choosing which option to apply depending on the observations y_t collected over the states of the process. In our model, an option is represented by a vector denoted $o_t \in \mathbb{R}^O$, O being the size of the options representation space. (ii) Choosing a new option o_t will then initialize the second component (or **actor model**). During the next time steps, the actor model will sequentially choose actions based on observations x_t and update its state until a new option is generated. (iii) The **acquisition model** denoted $\sigma_t \in \{0; 1\}$ will decide if the model has to acquire y_t or not.

To better understand this two-levels architecture, we provide the inference pseudo-code in Algorithm 2 and the architecture of BONN in Figure 1. We now describe the resulting components (the details are given in the Appendix):

Option model: The option model will be denoted f such that $f(y_t) = o_t$ generates an option o_t as a latent vector in a latent space $\mathbb{R}^O$, O being the dimension of this space. Note that the option model is a deterministic model where the option is computed based on the current observation using a neural network (see Appendix). Recurrent versions of the option model will be studied in a future work.

Actor Model: The state of the actor model is represented by a vector $z_t \in \mathbb{R}^Z$, Z being the size of the latent space of the actor model. At each time step, the distribution over the possible set of actions is computed by the function d such that $P(a_t|z_t) \approx d(z_t, a_t)$. Note that d is typically based on a soft-max function mapping action scores to action probabilities (see Appendix). If a new option o_t is computed by the option model, then the state z_t is re-initialized with $z_t = p(o_t, x_t)$. p is a *reset function* which aims at choosing the "initial" state of the actor for each new option. If

Algorithm 2 The pseudo code of the inference algorithm for the BONN model.

```
1: procedure INFERENCE($s_1$)                                        ▷ $s_1$ is the initial state
2:     initialize $z_0$ with the empty option= $(0, 0, ..., 0) \in \mathbb{R}^O$
3:     for $t = 1$ to $T$ do
4:         acquisition model: Draw $\sigma_t \sim h(z_{t-1}, a_{t-1}, x_t)$
5:         if $\sigma_t == 1$ then
6:             option level: Acquire $y_t$ and compute a new option $o_t = f(y_t)$
7:             actor level: Initialize the actor $z_t = r(o_t, x_t)$
8:         else
9:             actor level: Update the actor state $z_t = h(z_{t-1}, a_{t-1}, x_t)$
10:        end if
11:        actor level: Choose the action $a_t$ w.r.t $z_t$
12:        Execute the chosen action
13:    end for
14: end procedure
```

a new option is not generated, the actor state is updated with a classical recurrent mechanism i.e $z_{t+1} = g(z_t, a_t, x_{t+1})$

Acquisition Model: The acquisition model aims at deciding if a new option has to be generated. It is a stochastic process such that $\sigma_t = 1$ (new option) or $\sigma_t = 0$ (keep the same option) and is computed over the state of the actor and the new observation x_t: $P(\sigma_{t+1} = 1) = h(z_t, a_t, x_{t+1})$. In our case, this probability is based on a Bernoulli distribution over a sigmoid-based h function (see Appendix).

3.2 Budgeted Learning for Options Discovery

The way options emerge in a hierarchical reinforcement learning system has been the topic of many different works in both reinforcement learning and cognitive science. Most of these techniques associate the problem of option discovery with the problem of sub-goals discovery where different strategies are used to discover the sub-goals – see Botvinick et al. (2009) for a review on links between cognitive research and hierarchical reinforcement learning. The BONN model is based on a different approach, where we consider that the discovery of options will result in learning a good trade-off between policy efficiency and the *cognitive effort* generated by such a policy. The underlying idea is that a system will learn relevant options if these options allow to reduce the *cognitive effort* that is generated when solving the task, without decreasing the quality of the solution. Note that the reduction of the cognitive effort has already been studied in cognitive science (Kool & Botvinick, 2014), and very recently in the RL context (Bacon & Precup, 2015b) but defined differently.

Here, the cognitive effort is associated with the acquisition of the additional information y_t, this additional information (and its computation) being considered costly but crucial for discovering a good policy: an agent only using the observations x_t would be unable to solve the task, but using y_t at each time step would be "too expensive". The design choice of x_t and y_t is fundamental in the architecture, and is a distinct solution for bringing expert knowledge rather than explicitly defining sub-tasks.

Let us denote $C = \sum_{t=0}^{T-1} \sigma_t$ the acquisition cost for a particular episode: by reducing C, we will encourage the agent to learn relevant options that will be used during many time steps, the model extracting relevant sub-policies. 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 (Contardo et al., 2016; Dulac-Arnold et al., 2012). We define an augmented 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 task efficiency and the cognitive charge. The resulting discounted return denoted R_0^* will be used as the objective to maximize instead of the classical discounted return R_0, resulting in the following policy gradient update rule:

$$\pi \leftarrow \pi - \gamma \sum_{t=0}^{T-1} \left(\nabla_\pi \log P(a_t|z_t) + \nabla_\pi \log P(\sigma_t|z_{t-1}, a_{t-1}, x_t) \right) (R_t^* - b_t^*) \tag{4}$$

where γ is the learning rate. Note that this rule now updates both the probabilities of the chosen actions a_t, but also the probabilities of the σ_t that can be seen as *internal actions* and that decide if a new option has to be computed or not, b_t^* being the new resulting variance reduction term.

3.3 Discovering a discrete set of options

In the previous sections, we considered that the option o_t generated by the option model is a vector in a latent space $\mathbb{R}^O$. This is slightly different than the classical option definition which usually considers that an agent has a given "catalog" of possible sub-routines i.e the set of options is a finite discrete set. We propose here a variant of the model where the model learns a finite discrete set of options.

Let us denote K the (manually-fixed) number of options one wants to discover. Each option will be associated with a (learned) embedding denoted o^k. The *option model* will store the different possible options and choose which one to use each time an option is required. In that case, the option model will be considered as a stochastic model able to sample one option index denoted i_t in $\{1, 2, ..., K\}$ by using a multinomial distribution on top of a softmax computation. In that case, as the option model computes some stochastic choices, the policy gradient update rule will integrate these additional internal actions with:

$$\pi \leftarrow \pi - \gamma \sum_{t=0}^{T-1} \left(\nabla \log P(a_t|z_t) + \nabla \log P(\sigma_t|z_{t-1}, a_{t-1}, x_t) + \nabla \log P(i_t|y_t) \right) (R_t^* - b_t) \tag{5}$$

By considering that $P(i_t|y_t)$ is computed based on a softmax over a scoring function $P(i_t|y_t) \approx \ell(o_{i_t}, y_t)$ where ℓ is a differentiable function, the learning will update both the ℓ function and the options embedding o_k.

4 Experiments

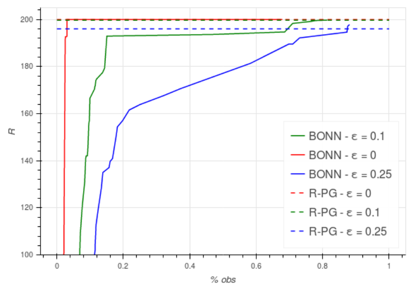
The complete details of the architecture used for the experiments are provided in the Appendix. We have tested this architecture on 3 different types of environments and compared it to a **Recurrent Policy Gradient algorithm** using a GRU-based neural network (R-PG):

CartPole: This is the classical cart-pole environment as implemented in the OpenAI Gym[4] platform where observations are $(position, angle, speed, angularspeed)$, and actions are $right$ or $left$. The reward is +1 for every time step without failure. For BONN, the observation is only used by the option model i.e $y_t = (position, angle, speed, angularspeed)$, the actor model receiving an empty observation $x_t = \emptyset$ at each time step.

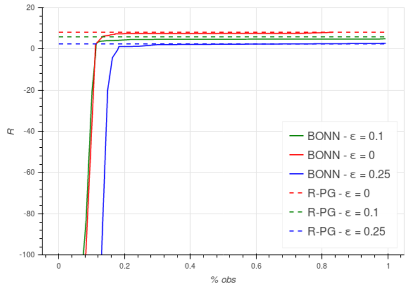
Lunar Lander: This environment corresponds to the Lunar Lander environment proposed in OpenAI Gym where observations describe the position, velocity, angle of the agent and if he is in contact with the ground or not, and actions are *do nothing, fire left engine, fire main engine, fire right engine*. 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. As for the cart-pole, the observation is only acquired by the option model, the actor model receiving an empty observation $x_t = \emptyset$ at each time step.

Multi-room Maze: The Multi-room Maze corresponds to a maze composed of $k \times k$ rooms ($k = 2$ or $k = 3$), with doors between them (see Figure 3). 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: $(up, down, left, right)$. We consider two variants: $MAZE_1$ where $x_t = \emptyset$ is the empty observation and the agent must learn when to acquired the more informative observation y_t which contains the observed doors, the agent position and the goal

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

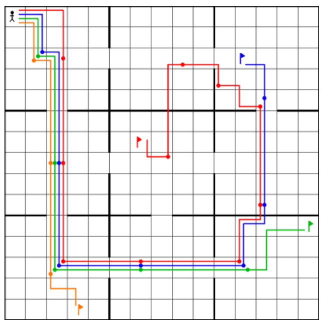


(a)

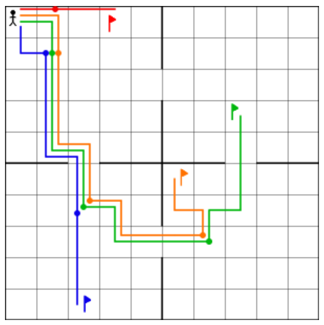


(b)

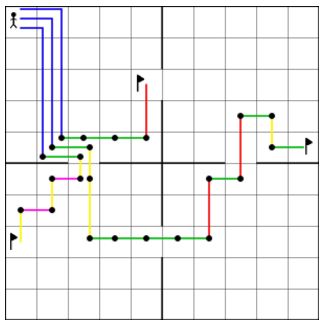
Figure 2: Cost/reward curves for cart-pole (a) and $MAZE_2$ (b) with different levels of stochasticity. The X-axis corresponds to the ratio of options used in each episode (100% means that the agent observes y_t and computes a new option at each time step). The Y-axis corresponds to the reward R obtained on the task. The dashed lines are the R-PG performance. Note that the R-PG performance is obtained without options (i.e using all the information available at each time step)



(a)



(b)



(c)

Figure 3: (a) (b) Examples of trajectories generated by the agent. Each point corresponds to a position where the agent decides to acquire y_t and generates a new option. (c) Trajectories generated with the D-BONN model where $K = 9$.

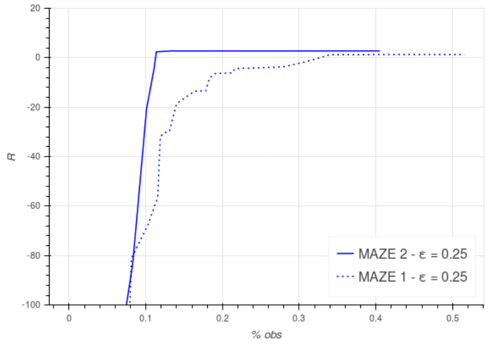
position if the goal is in the same room than the agent (i.e the agent only observes the current room). In the $MAZE_2$ world, x_t is the agent position in the room, while y_t corresponds to the description of the room and the goal position if the goal is in the same room (i.e contrary to $MAZE_1$, the agent always observes his position). The R-PG baseline has access to all these information (doors, position, goal) at each time step. Note that this environment is much more difficult than other 4-rooms problems (introduced by Sutton et al. (1999)) in others RL works, where there is only one or two goal position(s), and that, in a more realistic way, the agent only observes the room he is in.

For all the environments, we consider different levels of stochasticity ϵ such that the movement of the agent can fail with probability ϵ, in which case a random transition is applied.

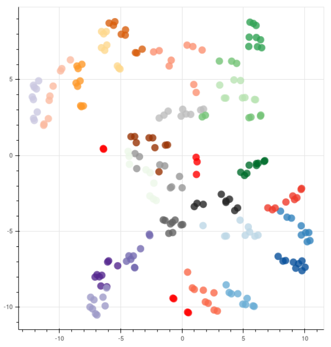
the action chosen by the agent is applied with probability $1-\epsilon$ while a random action is chosen with probability ϵ. The higher epsilon is, the more the environment is stochastic.

4.1 Quantitative Analysis

We illustrate the quality of BONN with cost/reward curves (see Figure 2) where the X-axis corresponds to the number of times an option is computed (normalized w.r.t the size of the episode) while the Y-axis corresponds to the overall reward $R = \sum r(s_t, a_t)$ obtained on each task, for different levels of stochasticity ϵ. Note that cost/reward curves are generated by computing the Pareto front over all the learned models at different cost levels λ. These curves have been obtained by first learn-



(a)



(b)

Figure 4: (a) Cost/reward curve for $MAZE_1$ and $MAZE_2$ with a stochasticity level of $\epsilon = 0.25$. (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.

ing our model with a zero cost $\lambda = 0$ and then by progressively increasing this cost, forcing the model to acquire y_t less frequently and to discover options[5].

First, one can see that even at low cost values (with only a few options computed), the BONN model is able to keep the same performance than the R-PG model, even if R-PG uses all the information contained in both x_t and y_t at each time step. Some specific cost/reward values are given in Table 1 for different environments and different values of λ, confirming that BONN is able to keep a high performance level while discovering relevant options. Note that if the cost of computing a new option is too expensive, the BONN model is not able to find a good policy since it is not allowed to switch between options.

We can also see that the obtained reward decreases when the environments are more stochastic, which seems intuitive since stochasticity makes the tasks harder. Figure 4a compares the results obtained on the $MAZE_1$ environment and the $MAZE_2$ environment when $\lambda = 0.25$. We note that the drop of performance in $MAZE_2$ happens at a lower cost than the one in $MAZE_1$. Indeed, in $MAZE_2$, the agent has access to its position at each time step and is more able to "compensate" the stochasticiy of the environment than in the $MAZE_1$ case, where the position is only available through y_t.

Figures 3b and 3a illustrates trajectories generated by the agent in the $MAZE_2$ 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 (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 cannot find the shortest path to the goal's room because, having no information about the position of the goal in another room, it learns to explore the maze in a "particular" order until reaching the goal's room. We have 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 where the goals are in similar areas. We can for example see that all green options are close, and it shows that the latent option space has captured a particular structure. Analyzing this latent structure will be the topic of a future research.

The D-BONN model has been experimented on the $MAZE_1$ 2×2, and an example of generated trajectories is given in Figure 3c. Each color corresponds to one of the learned discrete options. One can see that the model is still able to learn a good policy, but the constraint over the fixed number of discrete options clearly decreases the quality of the obtained policy. It seems thus more interesting to use continuous options instead of discrete ones, the continuous options being regrouped in smooth clusters as illustrated in Figure 4b.

[5]Learning separate models for many λ values is time-consuming and does not significantly improve the obtained results

		$\epsilon = 0$		$\epsilon = 0.25$	
		R	%obs	R	%obs
Cartpole	R-PG	200	1	196.02	1
	BONN $\lambda = 0.5$	199.76	0.06	181.65	0.26
	BONN $\lambda = 1$	190.346	0.05	172.23	0.20
$MAZE_2$ 3x3	R-PG	-5.86	1	-15.82	1
	BONN $\lambda = 3$	-5.67	0.19	-27.11	0.16
Lunar Lander	R-PG	227.35	1	109.31	1
	BONN $\lambda = 0.5$	221.24	0.16	91.68	0.07
	BONN $\lambda = 5$	210.51	0.06	90.41	0.04

Table 1: Cost/reward values for the different environments, at different cost levels λ and different stochasticiy levels ϵ

5 Related Work

Hierarchical Reinforcement Learning (Dayan & Hinton, 1993; Dietterich, 1998; Parr & Russell, 1998) has been the surge of many different works during the last decade because it is considered as one solution to solve long-range planning tasks and allows to transfer knowledge between tasks. Many different models have been proposed where subtasks are *a priori* known like Dietterich (1998) which proposes the MAXQ method. The concept of option has been introduced by Sutton et al. (1999). 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. Other works defines hierarchical policies based on different levels of observations: the Abstract Hidden Markov Model (Bui et al., 2002) is based on discrete options defined on each space region, while in Heess et al. (2016) the architecture uses a low-level controller that as only access to the proprioceptive information and a high-level controller has access to all observations.

The concept of options is at the core of many recent articles, for example in Kulkarni et al. (2016), the Deep Q-Learning framework is extended 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 option framework, Daniel et al. (2016) learns options (both policies and termination probabilities) without supervision using the Expectation Maximization algorithm. More recently, Bacon & Precup (2015a) does the same with the option-critic architecture, close to an actor-critic algorithm where options are discrete. They used a similar experiment to ours with four rooms but only one (fixed) goal ; learning both option policies and termination functions, the model converges to an option in the first room followed by a second one in rooms 2 and 3. The closest work to our seems to be Bacon & Precup (2015b) but the model is also based on a discrete set of options in the POMDP framework. Note that this article also introduces the cognitive effort concept. Some models are focused on the problem of learning macro-actions (Hauskrecht et al., 1998; Mnih et al., 2016). In that case a given state is mapped to a sequence of actions (i.e macro-actions), similar than when having x_t empty in the BONN-model. But macro-actions are more restricted than options since the sequence of actions is fixed.

The main difference between these works and the BONN architecture is that, in the case of BONN, options are latent vectors and the model is able to learn a manifold of possible options – even if a discrete version has also been proposed with less convincing performances.

Outside reinforcement learning, our work is also in relation with the Hierarchical Multiscale Recurrent Neural Networks (Chung et al., 2016) that discover hierarchical structure in sequences.

6 Conclusion and Perspectives

We have proposed a new model for learning options in POMDP where the agent can choose to acquire a more informative observation at each time step. The model is learned in a budgeted learning setting where the acquisition of an 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, and we have also proposed a discrete version of BONN where the number of options is kept constant. Experimental results show that the model is able to extract relevant options in complex environments. This work opens different research directions. One is to study if BONN can be applied in multi-task reinforcement learning problems (the environment $MAZE$, since the goal position is randomly chosen at each episode, can be seen as a particular simple multitask RL 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.

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

Pierre-Luc Bacon and Doina Precup. The option-critic architecture. In *NIPS Deep Reinforcement Learning Workshop*, 2015a.

Pierre-Luc Bacon and Doina Precup. Learning with options: Just deliberate and relax. 2015b.

Matthew Botvinick, Yael Niv, and Andrew C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement-learning perspective. *cognition*, 113.3, 2009.

Hung Hai Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Gabriella Contardo, Ludovic Denoyer, and Thierry Artières. Recurrent neural networks for adaptive feature acquisition. In *International Conference on Neural Information Processing*, pp. 591–599. Springer International Publishing, 2016.

Christian Daniel, Herke van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. 2016.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–271. Morgan Kaufmann Publishers, 1993.

Thomas G Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, pp. 118–126. Citeseer, 1998.

Gabriel Dulac-Arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. Sequential approaches for learning datum-wise sparse representations. *Machine Learning*, 89(1-2):87–122, 2012.

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 220–229. Morgan Kaufmann Publishers Inc., 1998.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Wouter Kool and Matthew Botvinick. A labor/leisure tradeoff in cognitive control. *Journal of Experimental Psychology: General*, 143(1):131, 2014.

Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, pp. 1–9, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, Koray Kavukcuoglu, et al. Strategic attentive writer for learning macro-actions. *arXiv preprint arXiv:1606.04695*, 2016.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pp. 1043–1049, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*, 2015.

Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.

ARCHITECTURE DETAILS

For all experiments, we used the ADAM optimizer (Kingma & Ba, 2014) with gradient clipping[6]

Option model: The option $o_t \in \mathbb{R}^O$ is generated by: $o_t = f(y_t) = \mathrm{relu}(W_o y_t)$ where W_o is a matrix of parameters.

Actor model: The state of the actor is $z_t \in \mathbb{R}^Z$ computed by:

- If a new option o_t is generated:
$$z_t = r(o_t, x_t) = \tanh(W_{zo}\,\mathrm{concat}(o_t, x_t)) \tag{6}$$
where W_{zo} is a matrix of parameters.
- else, the new state z_{t+1} is compute by a gated recurrent unit (GRU):
$$r = \mathrm{sigmoid}(W_r x_{t+1} + Y_a a_t + U_r z_t) \tag{7}$$
$$u = \mathrm{sigmoid}(W_u x_{t+1} + Y_u a_t + U_u z_t) \tag{8}$$
$$c = \tanh(W_c x_{t+1} + Y_c a_t + U_c(r \cdot z_t)) \tag{9}$$
$$z_{t+1} = u z_t + (1-u)c \tag{10}$$
where $\cdot$ is an element-wise multiplication and $W_.$, $Y_.$ and $U_.$ are matrices of parameters.
- in both cases, the distribution over the set of actions is compute by a softmax function on $d(z_t) = W_d z_t$ where W_d is a matrix of parameters.

Acquisition model: At time $t+1$, the probability to compute a new option is $h(z_t, a_t, x_{t+1}) = \mathrm{sigmoid}(U_h z_t + Y_h a_t + W_h x_{t+1})$ where U_h,Y_h and W_h are a matrices of parameters.

MAZE environment details: The position of the agent is given by a vector of length 5×5 with zeros everywhere and a one corresponding to its position in the room. The doors are encoded with a vector of length 4 (0 if no door, 1 if a door), and the goal (if present in the same room than the agent) is also encoded with a vector of length 5×5 (with only 0 if the goal is in another room).

Experiments: The only differences between experiments are the dimensions O and Z, and some hyper-parameters. The details of the tested values are given in Table 2.

	cartpole	lunar lander	maze1	maze2
dim of option space O	5	10	20	10
dim of x_t representation	$\emptyset$	$\emptyset$	$\emptyset$	10
size of actor state Z	5	10	20	10

Table 2: Values of parameters for the BONN architecture. Note that the values for $MAZE_1$ and $MAZE_2$ are both for the 2×2 maze and the 3×3 maze.

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