Options Discovery with Budgeted Reinforcement Learning

Aurélia Léon and Ludovic Denoyer
Sorbonne Universités,
UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
{aurelia.leon,ludovic.denoyer}@lip6.fr

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 Option Neural Network model (BONN), a hierarchical recurrent neural network architecture that learns latent options as continuous vectors. With respect to existing approaches, BONN 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.

1 Introduction

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 [Botvinick et al., 2009]. 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 [Dayan and Hinton, 1993; Dietterich, 1998; Parr and Russell, 1998], motivated by the idea that it makes the discovery of complex skills easier. One line of approaches consists in modeling sub-tasks through options [Sutton et al., 1999], 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 rarely addressed: the hierarchical structure in the majority of existing models has to be manually constrained, e.g. by predefining possible sub-goals [Kulkarni et al., 2016]. 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.

Recent studies in neurosciences suggest that a hierarchy can emerge from habits [Dezfouli and Balleine, 2013]. Indeed, [Keramati et al., 2011] and [Dezfouli and Balleine, 2012] 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.

Based on this idea, we propose the BONN architecture (Budgeted Options 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. $y_t$), BONN will learn a hierarchical structure where the resulting policy is a sequence of low-cost sub-policies. In that setting, we show that the structure of the resulting policy can be seen as a sequence of intrinsic options [Gregor et al., 2016], i.e. vectors in latent space.

The contributions of the paper are threefold:

- We propose a new assumption about options 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 cognitive cost.

1In that case, the cognitive cost is the time the animal spends to decide which action to take.
The closest architecture to BONN is the Hierarchical Multi-
notations and discussed in Section 5. At last, Section 6 concludes and
budgeted problem in Section 4. Experiments are presented
to introduce the BONN model and the learning algorithm for the
for recurrent policy gradient methods in Section 3. We in-
stantly show the ability of our approach to discover rel-
cative skills is at the core of many recent articles. For
options), and a termination function which defines the prob-
assumed that subtasks are a priori known, e.g., the MAXQ method in [Dietterich,
high-level controller has access to all observations.
Hauskrecht et al., 1999] is based on discrete options defined
reward. Some models focus on the problem of learning macro-
a cost. [Florensa et al., 2016] use a similar idea of factoring the state
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 [Dietterich, 1998]. The concept of option is 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. The concept of options is at the core of many recent articles. For example, in [Kulkarni et al., 2016], the authors 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, [Daniel et al., 2016] and [Bacon and Precup, 2015b] discover
options (both internal policies and the policy over options) without supervision, using respectively the Expectation Max-
immization algorithm and the option-critic architecture. Our
work, the concept of cognitive effort was introduced in [Bacon and Precup, 2015a] (but with discrete options), while intrinsic options, i.e., options as latent vectors, where used in [Gregor et al., 2016].

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 ob-
ervation from the environment at every time step. [Hansen et al., 1996] propose a model that mixes open-loop and closed-loop control while considering that sensing incurs a cost. Some models focus 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. Another category of models divides the state space into several components. For instance, the Abstract Hidden Markov Model [Bui et al., 2002] is based on discrete options defined on each space region. [Heess et al., 2016] use a low-level controller that has only access to the proprioceptive information, and a high-level controller has access to all observations. [Florensa et al., 2016] use a similar idea of factoring the state space into two components, and learn a stochastic neural net-
work for the high-level controller. The blind setting of the
BONN model, described in Section 5.2, is similar to (stochastic) macro-actions, but open-loop policies are rather limited in complex environments. The general BONN architecture is more comparable to works using two different observations, however, those models do not learn when to use the high-level controller.

3 Background

3.1 (PO-) Markov Decision Processes and
Reinforcement Learning

Let us denote a Markov Decision Process (MDP) as a set of states \( S \), a discrete set of possible actions \( A \), 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 \( \bar{P}_t \) 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.

3.2 Learning with Recurrent Policy Gradient

Let us denote \( \gamma \in [0, 1] \) the discount factor, and

\[
R_t = \sum_{k=1}^{T-1} \gamma^{k-1} r(s_k, a_k)
\]

the discounted sum of rewards (or dis-
count return) at time \( t \), corresponding to the trajectory \((s_t, a_t, s_{t+1}, a_{t+1}, \ldots, s_T)\) with \( T \) the size of the trajectories sampled by the policy. 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 \( \pi^* \) is computed by maximizing the expected discounted return \( J(\pi) \):

\[
J(\pi) = \mathbb{E}_{s_0 \sim \text{Prior}, a_0, \ldots, a_{T-1} \sim \pi} [R_0]
\]

where \( s_0 \) is sampled following \( P_I \) and the actions are sampled based on \( \pi \).

Different learning algorithms aim at maximizing \( J(\pi) \). In the case of policy gradient techniques, if we consider that, for sake of simplicity, \( \pi \) also denotes the set of parameters of the policy, the gradient of the objective can be approximated with:

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

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).

4 Budgeted Option Neural Network

4.1 The BONN 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 BONN 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 \((x_t)\) can decide to consult its map or GPS \((y_t)\); a virtual agent taking decisions in a virtual world \((x_t)\) 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 basically decide whether it wants to use the current observation, as in the goal directed vs habits paradigm [Keramati et al., 2011; Dezfouli and Balleine, 2012].

The structure of BONN 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 BONN 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 BONN architecture.

Algorithm 1 The pseudo code of the inference algorithm for the BONN model.

1: procedure INFERNCES\((s_0)\) \hspace{2em} \( s_0 \) is the initial state
2: \hspace{1em} initialize \( o_{-1} \) and \( h_{-1} \)
3: for \( t = 0 \) to \( T \) do
4: \hspace{1em} Acquire \( x_t \)
5: \hspace{1em} acquisition model: Draw \( \sigma_t \in \{0, 1\} \)
6: \hspace{1em} if \( \sigma_t == 1 \) then
7: \hspace{2em} option level: Acquire \( y_t \) and update the option state \( o_t \)
8: \hspace{2em} actor level: Initialize the actor state \( h_t \)
9: \hspace{1em} else
10: \hspace{2em} actor level: Update the actor state \( h_t \)
11: \hspace{1em} end if
12: \hspace{2em} actor level: Choose the action \( a_t \) w.r.t \( h_t \)
13: \hspace{2em} Execute the chosen action
14: end for
15: end procedure

Figure 2: The BONN 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 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.\(^1\)

**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 last computed option before time step $t$. $gru_{opt}$ represents a GRU cell \cite{Cho2014}.

**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 $\sigma$ and the availability of the high-level observation $y_t$:

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

The next action $a_t$ is then drawn from the distribution $softmax(f_{act}(h_t))$.\(^2\)

**Options and BONN:** When a new high-level observation is acquired i.e $\sigma_t = 1$, the option state $o_t$ is updated based on the current observations $x_t$ and $y_t$. Then, the policy will behave as a classical recurrent policy until a next 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 $\sigma_t$ represents what is usually called the termination function in option settings. In BONN, 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.

### 4.2 Budgeted Learning for Options Discovery

Inspired by cognitive sciences \cite{Kool2014}, BONN considers that discovering options 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 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, BONN 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 \cite{Contardo2016}.

\[ R^*(s_t, a_t, \sigma_t) = R(s_t, a_t) - \lambda \sigma_t \]  

where $\lambda$ 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, resulting in the following policy gradient update rule:

\[ \pi \leftarrow \pi - \gamma \sum_{t=0}^{T-1} (\nabla_\pi \log P(a_t|s_t) + \nabla_\pi \log P(\sigma_t|s_{t-1}, a_{t-1}, x_t)) (R^*_t - b_t) \]  

where $\gamma$ is the learning rate. Note that this rule now updates both the probabilities of the chosen actions $a_t$ and the probabilities of the $\sigma_t$ that can be seen as internal actions and that decide if a new option has to be computed or not. $b_t$ is the new resulting variance reduction term as defined in \cite{Wierstra2010}.

### 4.3 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. 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:

\[ \pi \leftarrow \pi - \gamma \sum_{t=0}^{T-1} (\nabla_\pi \log P(a_t|s_t) + \nabla_\pi \log P(\sigma_t|s_{t-1}, a_{t-1}, x_t)) + \nabla_\pi \log P(i_t|y_t)) (R^*_t - b_t) \]

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 $\ell$ is a differentiable function, the learning will update both the $\ell$ function and the options embedding $o_k$.

### 5 Experiments

#### 5.1 Experimental setting

For all experiments, we used the ADAM optimizer \cite{Kingma2014} with gradient clipping\(^3\). The learning rates were optimized by grid-search.

\(^{-}^{-}^{-}\)An open-source version of the model is available at https://github.com/aurelijae/BONN-model.
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) [Cho et al., 2014].

### 5.2 Blind Setting

Given a POMDP (or MDP), the easiest way to design the two observations $x_t$ and $y_t$ needed for the BONN 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 BONN in this setting: (i) **CartPole**: This is the classical cart-pole environment as implemented in the OpenAI Gym platform [Brockman et al., 2016] 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 $N_{gru} = 5$. (ii) **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 $N_{gru} = 10$. (iii) **$k \times k$-rooms**: This environment corresponds to a maze composed of $k \times k$ rooms with doors between them (see Figure 3a). 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-fold problems (introduced by Sutton et al., 1999). 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 $N_{gru} = 10$.

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 BONN 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 BONN learns to use $y_t$ only from time to time.

We illustrate the quality of BONN in Table 1. 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 $\epsilon$: in that case, a random transition is applied. In deterministic environments, we can see that the BONN 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. However, in stochastic environments, observations are used more often and even then the performances degrade much more. Indeed, due to the stochasticity, it is not possible for the agent to deduce its position using only the chosen action (due to failure of actions, 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.

### 5.3 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 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

---

Table 1: Cost/reward values for the different environments, at different cost levels $\lambda$ and different stochasticity levels $\epsilon$.

| Environment | $\epsilon = 0$ | $\epsilon = 0.25$ |
|-------------|----------------|------------------|
| Cartpole    | $x$ | $R$ | $%obs$ | $x$ | $R$ | $%obs$ |
| R-PG        | $\lambda = 0.5$ | 199.7 0.06 | 181.6 0.26 | 199.7 0.06 | 181.6 0.26 |
| BONN $\lambda = 1$ | 190.3 0.05 | 172.2 0.20 | 190.3 0.05 | 172.2 0.20 |
| R-PG        | $\lambda = 1$ | -7.4 0.36 | -16.3 0.61 | -7.4 0.36 | -16.3 0.61 |
| Lunar Lander| $\lambda = 0.5$ | 221.2 0.16 | 91.6 0.07 | 221.2 0.16 | 91.6 0.07 |
| BONN $\lambda = 5$ | 210.5 0.06 | 90.4 0.04 | 210.5 0.06 | 90.4 0.04 |

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

---

https://gym.openai.com/
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 $N_{\text{gru}} = 10$.

In a stochastic environment ($\epsilon = 0.25$), BONN 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. 5 the rewards w.r.t. cost curves obtained by computing the Pareto front over BONN models with different cost levels $\lambda$. We note that the drop of performance in $2 \times 2$-rooms with $x_t = \text{position}$ happens at a lower cost than the one in $2 \times 2$-rooms environment with blind setting. Indeed, with $x_t = \text{position}$, 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 5.4.

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.

### 5.4 Analysis of option discovered

Figures 4a illustrates trajectories generated by the agent in the $3 \times 3$-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. 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.

The D-BONN model has been experimented on the $2 \times 2$-rooms environment, and an example of generated trajectories is given in Fig. 5. 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.

### 5.5 Instructions as options

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 $N_{\text{gru}} = 5$ (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. 6. Figure 6b 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
Figure 6: 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.

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 6a), 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 BONN model seems to be an original way to study the Model Free/Model-based paradigm proposed in neuroscience [Gläscher et al., 2010].

6 Conclusion and Perspectives

We proposed a new model for learning options in POMDP 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 BONN 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.

Acknowledgments

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.
