Learning Purposeful Behaviour in the Absence of Rewards

Marlos C. Machado  
MACHADO@UALBERTA.CA  
Michaël Bowling  
MBOWLING@UALBERTA.CA  
Department of Computing Science, University of Alberta, Canada

Abstract

Artificial intelligence is commonly defined as the ability to achieve goals in the world. In the reinforcement learning framework, goals are encoded as reward functions that guide agent behaviour, and the sum of observed rewards provide a notion of progress. However, some domains have no such reward signal, or have a reward signal so sparse as to appear absent. Without reward feedback, agent behaviour is typically random, often dithering aimlessly and lacking intentionality. In this paper we present an algorithm capable of learning purposeful behaviour in the absence of rewards. The algorithm proceeds by constructing temporally extended actions (options), through the identification of purposes that are “just out of reach” of the agent’s current behaviour. These purposes establish intrinsic goals for the agent to learn, ultimately resulting in a suite of behaviours that encourage the agent to visit different parts of the state space. Moreover, the approach is particularly suited for settings where rewards are very sparse, and such behaviours can help in the exploration of the environment until reward is observed.

1. Introduction

Reinforcement learning (RL) has been successful in generating agents capable of intelligent behaviour in initially unknown environments; with accomplishments such as surpassing human-level performance in Backgammon (Tesauro, 1995), helicopter flight (Ng et al., 2004), and general competency in dozens of Atari 2600 games (Mnih et al., 2015). Such successes are achieved by algorithms that maximize the expected cumulative sum of rewards, which can be seen as a measure of progress towards the desired goal. The goal is sometimes easily defined through rewards, such as $\pm 1$ signals encoding win/loss in games or $-1$ signals informing the agent something undesirable occurred (e.g., a robot bumping into a wall).

We are interested in the setting where the reward signal is uninformative or even absent. In the uninformative setting, the paucity of reward usually leads to dithering typical of $\epsilon$-greedy exploration approaches. This effect is particularly pronounced when the agent operates at a fine time scale, as is common of video game platforms (Bellemare et al., 2013). In the complete absence of reward, it is unclear what intelligent behaviour should even constitute. Although intrinsic motivation-based approaches (Oudeyer et al., 2007; Barto, 2013) offer a solution in the form of an intrinsic reward signal, here we instead consider the notion of a purposeful agent: one that can commit to a behaviour for an extended period of time.

To construct purposeful agents, we appeal to the options framework (Sutton et al., 1999). Options extend the mathematical framework of reinforcement learning and Markov decision processes (MDPs) to allow agents to take temporally extended actions to accomplish subgoals. While this extension is an extremely powerful idea for allowing reasoning at different levels of abstraction, automatically discovering options (e.g., by identifying subgoals) is an open problem in the literature. Generally, options are designed by practitioners who identify meaningful subgoals that can be used as stepping-stones to solve a complex task. Besides using options to divide a task into subgoals, we advocate one can also use options to add decisiveness to agents in an environment in which rewards are not available, and that this is a better choice than aimless exploration.

In this paper we introduce an algorithm capable of learning purposeful behaviour in the absence of rewards. Our approach discovers options by identifying purposes that are achievable by the agent. These purposes are turned into intrinsic subgoals through to an intrinsic reward function, resulting in a suite of behaviours that encourage the agent to visit different parts of the state space. These options are particularly useful in the absence of rewards. As an example, when the agent observes a change in the environment through its feature representation, it tries to learn a policy...
Learning Purposeful Behaviour in the Absence of Rewards

capable of reproducing that change. Also, when such an option is added to the agent’s action set, the agent now can move farther in the state-space, with some events that were rare now becoming frequent and events that were “impossible” now becoming “just” infrequent.

In this early paper we introduce the main ideas that underly our algorithm, including concepts such as “purpose”. We also provide an algorithm for options discovery with linear function approximation, in contrast to most approaches for option discovery that rely on tabular representations. Additionally, we show how our algorithm is able reach farther points from the starting state giving evidence of discovered intentionality.

2. Background

In this section we introduce the reinforcement learning (RL) problem setting and the options framework. We also discuss the problem of option discovery and some approaches that try to address it. As a convention, we indicate random variables by capital letters (e.g., \( S_t, R_t \)), vectors by bold letters (e.g., \( \theta \)), functions by lowercase letters (e.g., \( v \)), and sets by calligraphic font (e.g., \( S, A \)).

2.1. Reinforcement Learning and Options

In the RL framework (Sutton & Barto, 1998; Szepesvári, 2010) an agent aims at maximizing some notion of cumulative reward by taking actions in an environment; these actions may affect the next state the agent will be as well as all subsequent rewards it will experience. It is generally assumed the tasks of interest satisfy the Markov property, being called Markov decision processes (MDPs). An MDP is formally defined as a 5-tuple \( \langle S, A, r, p, \gamma \rangle \). At time \( t \) the agent is in the state \( s_t \in S \) where it takes an action \( a_t \in A \) that leads to the next state \( s_{t+1} \in S \) according to the transition probability kernel \( p(s' | s, a) \), which encodes \( P_r(S_{t+1} = s' | S_t = s, A_t = a) \). The agent also observes a reward \( R_{t+1} \sim r(s, a, s') \). The agent’s goal is to learn a policy \( \pi : S \times A \rightarrow [0, 1] \) that maximizes the expected discounted return \( G_t = \mathbb{E}_\pi \left[ \sum_{k=0}^\infty \gamma^k R_{t+k+1} | s_t \right] \), where \( \gamma \in [0, 1) \) is known as the discount factor. We are interested in settings where the reward signal \( R_t \) is uniform across the environment.

When learning to maximize \( G_t \) it is common to learn an action-value function defined as \( q_\pi (s, a) \approx \mathbb{E}_{\pi} [G_t | S_t = s, A_t = a] \). However, in large problems it may be infeasible to learn \( q_\pi \) exactly for each state-action pair. To tackle this issue agents often learn an approximate value function: \( q_\theta (s, a) \approx \theta^\top \phi(s, a) \). A common approach is to approximate these values using linear function approximation where \( q_\theta (s, a; \theta) \approx \theta^\top \phi(s, a) \), in which \( \theta \) denotes the vector of weights and \( \phi(s, a) \) denotes a static feature representation of the state \( s \) when taking action \( a \). This can also be done through non-linear function approximation methods such as neural networks (e.g., Tesator, 1995, Mnih et al., 2015). Note that generally \( \theta \) has much smaller number of parameters than the number of states in \( S \).

The standard RL framework is focused on MDPs, in which actions last a single time step. Nevertheless, it is convenient to have agents encoding higher levels of abstraction, which also facilitate the learning process if properly defined (Dietterich, 1998; Sutton et al., 1999). Sutton et al. extended the RL framework by introducing temporally extended actions called options. Intuitively, options are higher-level actions that are extended through several time steps. Formally, an option \( o \) is defined as 3-tuple \( o = \langle I, \varpi, T \rangle \) where \( I \subseteq S \) denotes the initiation set, \( \varpi : A \times S \rightarrow [0, 1] \) denotes the option’s policy, and \( T \subseteq S \) denotes the termination set. After initiated, actions are selected according to \( \varpi \) until the agent reaches a state in \( T \). Originally, Sutton et al. defined a function \( \beta : S \rightarrow [0, 1] \) to encode the probability of an option terminating at a given state. In this paper we define \( \beta \) to be the characteristic function of the set \( T \): \( \beta(s) = 1 \) for all \( s \in T \) and \( \beta(s) = 0 \) for all \( s \notin T \), hence we overload the notation. Options generalize MDPs to semi-Markov decision processes (SMDPs) in which actions take variable amounts of time. Options are also useful when addressing the problem of exploration because they can move agents farther in the state-space.

2.2. Option Discovery

The potential of options to dramatically affect learning by improving exploration is well known (e.g., McGovern & Sutton, 1998; McGovern & Barto, 2001; Kulkarni et al., 2016). Nevertheless, most works that benefit from options do not discover them, but assume they are provided.

The works that investigate how to discover options can be clustered in three different categories. The most common approach is to try to identify subgoal states through some heuristic such as visitation frequency (McGovern & Barto, 2001), graph-related metrics such as betweenness (Şimşek & Barto, 2008), or graph partitioning metrics (Menache et al., 2002; Mannor et al., 2004; Șimşek et al., 2005). Some authors have also tackled the problem of option discovery by trying to identify common subpolicies (Thrun & Schwartz, 1994; Pickett & Barto, 2002), while others proposed methods based on the frequency of the change of state variables (Hengst, 2002).

The works on option discovery generally operate in a tabular setting where you can have states uniquely defined. Consequently, metrics such as frequency of visitation and transition graphs can be used for option discovery. Automatically discovering options in large state-spaces where function approximation is required is still a challenge. Our
work presents an approach for option discovery in settings with linear function approximation, which has a much larger applicability. Few works tackled option discovery with function approximation. Those that did generally simplified the problem with additional assumptions such as knowledge of subgoal states (Konidaris & Barreto, 2009) or that one can control the interface between regions of the MDP (Hengst, 2002).

The proposals of not depending on a reward signal to discover meaningful behaviour (Şimşek & Barto, 2004), and of looking at the different rates of changes in the agent’s feature representation (Hengst, 2002) are related to our work. This relationship will be clearer in the next section.

3. Option Discovery for Purposeful Agents

Approaches based on intrinsic motivation and novelty are some of the ways to circumvent the absence of rewards in the environment. Schmidhuber (2010) summarizes several works based on intrinsic motivation, which he defines as algorithms that maximize a reward signal derived from the agent’s learning progress. Lehman & Stanley (2011) have advocated that agents should drop feedback they receive from the environment even in more traditional settings such as search, maximizing novelty instead. Both ideas are related to our work. We discover options based on novelty assuming no extrinsic rewards are available (Lehman & Stanley, 2011). These options are based on a very loose notion of a model of the environment, aiming at learning how to change principal components of a compressed environment representation (Schmidhuber, 2010).

Our algorithm is based on four different concepts, namely: (i) storing the changes seen between two different time steps, (ii) clustering correlated changes in order to extract a purpose, (iii) learning policies capable of reproducing desired purposes, and (iv) transforming these policies into options that can be used to move the agent farther in the state space. After these steps a new set of options giving different purposes to the agent is discovered. These options guide the agent to different parts of the state space, which may lead to identifying new purposes. When such steps are used iteratively we create a self-reinforcing loop. We discuss each concept individually while introducing the algorithm. The algorithm we introduce uses a binary feature representation, but it is extensible to more general settings.

The agent initially follows some default policy (possible random) for a given number of time steps, using all actions available on its action set. While following such policy, at every time step the agent stores in a dataset \( D \) the difference between the feature representation of its current observation \( \phi(s_t) \) and the representation of its previous observation \( \phi(s_{t-1}) \), i.e.: \( D \leftarrow D \cup \{ \phi(s_t) - \phi(s_{t-1}) \} \). It is important to stress that while storing changes one can easily see those that stand out, such as features that rarely change. Storing the features is less informative than the current transition because it is harder for the agent to identify when a feature really changed. Moreover, storing differences allow us to clearly identify different directions in the change, something that is useful in the next steps.

After a pre-defined number of time steps the agent stops storing transitions to identify purposes in the observed behaviour. It clusters together features that change together, avoiding correlated changes to generate the same purpose. Formally, such step consists in reducing the dimensionality of \( D \) through singular value decomposition (SVD): \( D = U \Sigma V^\top \). The SVD generates a lower rank representation of the transitions stored in \( D \). Such low rank representation consists of eigenvalues and eigenvectors. The eigenvectors encode the principal components of \( D \) while the eigenvalues weight them according to how much that eigenvector is important to reconstruct \( D \). Each eigenvector can be seen as encoding a different purpose for the agent because all features that are somehow correlated are collapsed to a single eigenvector, explaining a direction of variation of the observed transitions. We call the eigenvectors obtained from the dataset of transitions eigenpurposes.

**Definition 3.1 (Eigenpurpose).** Given a matrix \( D \) of transitions where each row encodes the difference between two consecutive observations, i.e. \( \phi(s_t) - \phi(s_{t-1}) \), and having \( V_i \) denoting the \( i \)-th row of matrix \( V \), each eigenvector \( (V^*)_i \), obtained from the singular value decomposition traditionally defined as \( D = U \Sigma V^\top \) is called an eigenpurpose.

The following example provides an intuition about the importance of eigenpurposes. Imagine that an agent, by chance, leaves a building. By doing it the value of several of its features change (e.g. lighting, temperature, soil). When we collapse all these changes with the SVD, instead of having a feature encoding “temperature increase”, other encoding “change of lighting”, and so on, we have only an eigenpurpose encoding “outside the building”.

Once eigenpurposes have been identified, the agent learns policies capable of maximizing their occurrence. In order to learn a policy that maximizes an eigenpurpose we need to define a proper reward function. Such reward \( r_{i,t} \) is defined as the similarity between the observed transition and the eigenpurpose of interest \( e_i \), i.e.: \( r_{i,t} = e_i^\top (\phi(s_t) - \phi(s_{t-1})) \). Because SVD does not encode signs, we learn how to maximize eigenpurposes in both possible directions, i.e. the agent also learns a second policy that maximizes \(-r_{i,t}\). These policies are called eigenbehaviours.

**Definition 3.2 (Eigenbehaviour).** A policy \( \pi : S \to A \) is called an eigenbehaviour if it is the optimal policy that maximizes the occurrence of an eigenpurpose (Definition 3.1) in the original MDP augu-
mented by the action \( \bot \) that terminates the option; 
\( i.e. \pi(s) = \arg\max_a \max_{s,a} q_\pi(s,a) \) in the MDP 
\( (S, A \cup \{\bot\}, (V^\top)_j (\phi(s_t) - \phi(s_{t-1})), p, \gamma) \).

The algorithm used to learn eigenbehaviours is not pre-
defined, nor the order in which they are learned.

We can naturally construct an option from the learned
eigenbehaviour. To do so, we need to define the set of 
states for which the eigenbehaviour is effective (initiation 
set) and its termination condition. We define the initiation 
set of an option as the set of states \( s \) in which, after learn-
ing, \( q_\pi(s,a) > 0 \) for at least one action \( a \in A \). Intuitively 
this corresponds to every state in which the policy can still 
made progress towards the eigenpurpose. The set of terminal 
states for this option is the complement of the initiation 
set, \( i.e. S \setminus I \). Once such options are discovered we can 
add them to the agents action set, which allows the agent 
to repeat the described process with a policy that uses the 
discovered options to collect new data.

Notice that eigenvalues loosely encode how frequent each 
eigenpurpose was observed. Therefore, the eigenbe-
haviours corresponding to the lower eigenvalues encode 
purposes observed less frequently. Because of that, once 
an option for a “rare” purpose is discovered, this purpose 
will no longer be “rare” since a single decision (taking the 
option) is now capable of reproducing this rare event. We 
may also increase the likelihood of observing other un-
likely “purposes” since a single action now moves the agent 
much farther in the state-space. This can help agents to ex-
plor environments in which rewards are very sparse, guid-
ing the agent until a reward signal is observed.

The described algorithm is formally presented in Algo-
rithm 1. An additional detail not discussed yet is that one 
can decide to learn how to maximize only a subset of the 
discovered eigenpurposes. In this work we did not evalu-
ate the impact of different approaches. Here we propose a 
simple eigenvalue threshold \( \kappa \) that determines which eigen-
purposes are interesting. We select all eigenpurposes that 
have a correspondent eigenvalue greater than \( \kappa \), which we 
interpret as discarding noise.

### 4. Experimental Evaluation

We performed an empirical validation in a hand-crafted do-
main which allows us to clearly illustrate our algorithm’s 
features, namely:

- At each new iteration of the algorithm the discovered 
options become increasingly more complex.
- More complex options are able to move the agent far-
ther away in the state space.
- As the agent moves farther away with newly discov-
ered options, it observes new eigenpurposes, discovering 
more options, what creates a self-reinforcing loop.

We evaluated our algorithm in a random walk in a toy do-
main consisting of moving around a ring of length 4096 
with deterministic transitions. The agent starts at the \( x \) 
coordinate 0 and at every time step it chooses between go-
ing right or going left. We use linear function approxima-
tion with the two’s complement binary encoding as rep-
resentation (12 bits long). When the agent goes left in the 
state 0 it goes to the state \( -1 \) (0000 0000 0000\( _2 \) \rightarrow 
1111 1111 1111\( _2 \)). Similarly, going right in state 2047 transi-
tions to \( -2048 \). There are no rewards in this environment.

All the evaluations were made in the same setting. The 
selected “noise” parameter \( \kappa \) was set to 1 and the environ-
ment, when learning the eigenbehaviours, had a discount
Learning Purposeful Behaviour in the Absence of Rewards

<table>
<thead>
<tr>
<th>Observ.</th>
<th>Metric</th>
<th>Iter. 0</th>
<th>Iter. 1</th>
<th>Iter. 2</th>
<th>Iter. 3</th>
<th>Iter. 4</th>
<th>Iter. 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>Num. Options Discov.</td>
<td>- (-)</td>
<td>5.9 (5.5)</td>
<td>7.7 (10.3)</td>
<td>8.5 (8.8)</td>
<td>9.2 (12.6)</td>
<td>9.5 (11.4)</td>
</tr>
<tr>
<td></td>
<td>Avg. Opt. Length</td>
<td>- (-)</td>
<td>12.1 (1.1)</td>
<td>19.2 (2.1)</td>
<td>21.6 (2.0)</td>
<td>25.5 (2.0)</td>
<td>27.8 (1.7)</td>
</tr>
<tr>
<td></td>
<td>Max Dist. from Start</td>
<td>29.3 (18.2)</td>
<td>168.7 (222.5)</td>
<td>240.1 (287.3)</td>
<td>269.9 (311.5)</td>
<td>287.1 (436.9)</td>
<td>298.9 (320.8)</td>
</tr>
<tr>
<td>Partial</td>
<td>Num. Options Discov.</td>
<td>- (-)</td>
<td>3.5 (20.9)</td>
<td>5.2 (14.1)</td>
<td>6.2 (19.4)</td>
<td>6.6 (30.2)</td>
<td>6.8 (21.6)</td>
</tr>
<tr>
<td></td>
<td>Avg. Opt. Length</td>
<td>- (-)</td>
<td>20.4 (1.8)</td>
<td>30.5 (1.4)</td>
<td>33.5 (2.1)</td>
<td>35.9 (1.7)</td>
<td>37.7 (1.7)</td>
</tr>
<tr>
<td></td>
<td>Max Dist. from Start</td>
<td>29.3 (18.2)</td>
<td>212.8 (326.9)</td>
<td>314.9 (314.3)</td>
<td>301.8 (434.3)</td>
<td>352.4 (464.1)</td>
<td>301.1 (360.0)</td>
</tr>
</tbody>
</table>

Figure 1. Sample random walk using primitive actions and a random walk using the discovered options. Dashed vertical lines represent iteration boundaries.

Table 1. Characteristics of discovered options, per iteration, when compared to a random walk in a ring. Each number is the average of 30 runs and standard deviations are reported between parentheses. For details about the selected parameters c.f. text.

(a) Full observability
(b) Partial observability

factor \( \gamma = 0.99 \). The policies were obtained through value iteration with 100 iterations. Each round of our algorithm consisted of 1,000 time steps in which the agent collected transitions to discover eigenpurposes. We ran six rounds, with round zero having only primitive actions available. The agent’s default policy was to select uniformly random among the currently available actions, or options.

We first evaluated our results in agents with full observability, in which the agent perceives states as described above (Figure 1, left, and Table 1). By looking at the average length of the discovered options we see that options become increasingly complex with each iteration. This added complexity allows the agent to move to farther states, as evidenced by the increasing distance between farthest point and the agent’s starting state at that iteration (Max Dist. from Start). The improvement is particularly clear when comparing to a sample random walk on primitive actions (Figure 1). Note that, even though options constructed in later iterations still use only primitive actions, they present more complex behaviours. This is in contrast with typical option discovery approaches, which construct hierarchies of options.

We also evaluated a setting in which the agent had partial observability. In this setting the agent does not observe the three least significant bits encoding the state. This collapses several states together and makes it much harder for the agent to observe progress. Interestingly, the same behavioural pattern as in the full observability experiment emerges. Agents still come up with options of the type “flip the \( i \)-th bit” once they discover these purposes. The unique difference is that fewer options are discovered at each iteration, due to fewer observable eigenpurposes.

5. Conclusion

In this paper we introduced a new algorithm capable of discovering options without any feedback in the form of rewards from the environment. Our algorithm discovers options that reproduce purposes extracted from the states seen by an agent acting randomly. We presented experimental results showing how the discovered options greatly improve exploration by introducing decisiveness on the agents, avoiding the traditional aimless dithering. We also showed evidences that our approach may work well with partial observability.

As future work, we plan to investigate how this algorithm behaves in more complex environments, such as the Arcade Learning Environment (Bellemare et al., 2013). We can evaluate our algorithm on these large domains because it is amenable to function approximation, differently from most other approaches for option discovery. Naturally, we then have to be able to learn a policy, using the discovered options, to maximize the discounted sum of rewards. Some of our preliminary results using interrupting options do seem promising. However, when applying this algorithm to larger domains we face a challenge not discussed here: the exploding number of eigenpurposes (and consequently discovered options), indicating that proper sampling techniques must be further evaluated. Finally, it is important to have coevolving action and representation abstractions: higher levels of action abstraction should drive the agent to improve its representation of the world and once the agent has a better representation of the world, better action abstractions should become available. This is a topic that is not commonly investigated but that needs to be addressed in the future, maybe our algorithm can be the first step towards that direction.

Acknowledgements

The authors thank Richard S. Sutton and Marc G. Bellemare for insightful discussions that helped improve this work. This work was supported by grants from Alberta Innovates Technology Futures (AITF) and the Alberta Innovates Centre for Machine Learning (AICML).
References


