The Options framework for Temporal Abstraction

Erfan Seyedsalehi
seyederfan.seyedsalehi@mail.mcgill.ca

May 1, 2020

Abstract

The problem of Temporal Abstraction is one of the most pivotal research areas in Reinforcement Learning today. The ability of the intelligent agent to make decisions about its actions in different Temporal scales is key to making Artificial Intelligence that is capable of performing complex tasks the way humans are able to do. One of the most promising methods to achieve this is by using a framework called Options which are smaller policies that can be executed during smaller periods of time with specific subgoals. First, we formally define options and show how they can be incorporated in the context of Markov Decision Processes. Then, we will move on to using model-based planning methods and model-free learning methods with options. We consider methods on a macro scale where options will be treated as black boxes and a micro scale where their policies can be changed and learned. We will discuss the strengths and weaknesses of the option framework and finally we will introduce the state of the art option-critic framework which allows the whole policy over options and the options to be learned simultaneously in an end-to-end manner.
Keywords: *Reinforcement Learning; Temporal Abstraction; Options.*

1 Introduction

When we look at human intelligence, Solving almost every task and problem involves decision making at different time scales. As an example, a task like cooking dinner involves some macro-scale decisions like what ingredients should be used with very small scale micro-actions such as moving muscles around and all these decision making at different temporal scales are seamlessly blended into human behaviour. It would be quite intuitive if artificially intelligent agents acted in the same way. Temporal Abstraction has seen significant attention from the Artificial Intelligence community with the rise of Reinforcement Learning which is one of the most natural ways to study AI. There has been quite a few different approaches in the Reinforcement Learning community to the problem of Temporal Abstraction. The most successful of these are the MAXQ method (Dietterich (1999)) , FeUdal networks for Hierarchical Reinforcement Learning (Vezhnevets et al. (2017)) and options (R. Sutton, Precup, and Singh (1999)) which will be the focus of this project report.

In Reinforcement Learning, problems are formulated in the format of Markov Decision Processes which are discrete time stochastic control processes where an agent interacts with an environment through its actions which are mathematically formulated as a policy. The MDP has a set of states that it can be in at each time step and the probability of the next state for each time step depends on the agents action and also a transition probability. After the agent takes an action at each time step, it receives a feedback from the environment in the form of a reward and the goal of the agent is to maximize the cumulative reward over time.

In order to deal with the problem of temporal abstraction in Reinforcement Learning, some constraints on MDPs should be relaxed to include cases where each action could take several time steps to complete and therefore the rewards are not given per time step. To formulate this, Semi-Markov decision processes are used which are a special kind of MDP with some relaxed constraints. Since we are dealing with multiple policies which act in different temporal scales, both MDPs and SMDPs are used to model how the agent decides about its actions and also how the environment evolves. It should be noted that for the sake of simplicity we only consider policies that act at two temporal scales.

In this report, first the theoretical foundations of options in the context of Reinforcement Learning is introduced and developed. Later, familiar methods in Reinforcement Learning such as Model-based planning with Dynamic Programming based methods, Model-free Control and Evaluation methods which rely on bootstrapping and sampling are extended to cases where actions are taken at different temporal scales. We explore the interplay between MDP and SMDP formulation to deal with policies governing over options and also policies inside options. Next, some limitations and constraints are discussed for learning and planning with options and finally the Option-Critic method (Bacon, Harb, and Precup (2016)) is discussed which is a family of policy gradient based methods which represent the state of the art in Options.

2 The Reinforcement Learning Problem

Just like any Reinforcement Learning problem, we model the world using and MDP which dictates how the environment evolves with the agent's actions. At each time-step $t = 0, 1, 2, ...$ the agent observes the state, takes action a_t from the set of actions A and receives reward r_{t+1} and this cycles continues until the environment terminates. The agent's action a_t at each time-step and the environment's state s_t impacts the reward r_{t+1} and state at the next times-step s_{t+1} according to a probability distribution function $P_{s,a}^{r,s'}$ which is called the model. The goal of the agent is to maximize the sum of rewards through an episode of experience. Sometimes, future rewards are discounted by a discount factor of γ per time-step.

In order to achieve the goal of reward maximization, we need to learn a policy $\pi : S \times A \to [0, 1]$ that does that. We define state and state-action values $V(s)$ and $Q(s, a)$ for a given policy. They represent the expected cumulative reward if the agent starts from state s and follows policy π (and chooses action a in the case of $Q(s, a)$). We then define V^* and Q^* as value functions and action value functions corresponding to the optimal policy π^*. Using the Bellman equation, we can find a recursive formulation for all value and action value functions.

$$\begin{aligned}V^{\pi}(s) &= \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, \pi\}\\ &= \mathbb{E}\{r_{t+1} + \gamma V^{\pi}(s_{t+1})|s_t = s, \pi\}\\ &= \sum_{a\in A} \pi(s,a)[\mathbb{E}\{r|s_t = s, a_t = a\} + \gamma \sum_{s'} P^{s'}_{s,a} V^{\pi}(s')]\end{aligned}$$

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$\begin{aligned}&Q^{\pi}(s,a) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, a_t = a, \pi\}\\ &= \mathbb{E}\{r|s_t = s, a_t = a\} + \gamma \sum_{s'} P^{s'}_{s,a} \sum_{a'} \pi(s',a') Q^{\pi}(s',a')\end{aligned}$$

$$\begin{aligned}Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a)\\ &= \mathbb{E}\{r|s_t = s, a_t = a\} + \gamma \sum_{s'} P^{s'}_{s,a} \max_{a'} Q^*(s',a')\end{aligned}$$

Using the above equations, there are various ways to solve the problem and obtain the optimal policy. If the model of the environment is known, Dynamic Programming based methods such as Policy Iteration can be used to efficiently learn the optimal Action value function and derive the optimal policy from it. If the model is unknown, model-free methods which are based on sampling can be used to obtain estimated of Value and Action-value function. Even in these cases, Bellman equations are quite helpful as they allow for using bootstrapping to learn the Value and Action-value functions. They form the basis for all Temporal Difference learning based methods. Ofcourse all of these methods are only useful in cases where actions are chosen from a discrete set. If the actions are continuous or even come from a very large action set, then these methods are not helpful and policy gradient based methods should be used.

3 Formulating Options

Now we define the options. Each Option $o \in O$ consists of a policy $\pi : S \times A \to [0, 1]$ (which gives the probability of taking an action a at state s), an initiation set $I \subseteq S$ (which determines a subset of states that the options can start from) and a termination condition $\beta : S \to [0, 1]$ (which determines the probability that the the option terminates at state at each state s). For now, we assume Whenever an option starts, it keeps executing actions according to the policy π until it stochastically terminates according to $\beta(s)$. In order to achieve Temporal Abstraction we use a policy over options $\mu : S \times O \to [0, 1]$ which decides what option should be executed at state s according to the probability distribution $\mu(s, .)$.

From the standpoint of μ options o are black boxes and their policies π_o are unknown. In other words, from the standpoint of μ options are seen as regular actions which should be taken according to the policy μ with the difference that each option takes an arbitrarily long number of time steps to execute. Here is where the problem is formulated as a Semi-MDP because each action(option) can take multiple time steps. Still, Action-value functions can be defined for each state option pair. And, they can be used to learn the optimal policy over a set of given options.

$$Q^{\mu}(s,o) = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, o_t = o, \mu\} \tag{1}$$

It is useful to make some clarification on the Semi-MDP context. If we are trying to to learn a policy over options μ and each option policy π_o is considered fixed, the problem becomes a Semi-MDP. But, if the problem is to learn a policy over actions for a specific option, the problem is still an MDP (for the duration that the option is executing) and this is one of the core foundations of the theory of Options. Even in the context of Semi-MDPs, we can still use bellman equations to obtain simpler mathematical formulations for value functions.

$$\begin{aligned}
V^{\mu}(s) &= \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k})|s_t = s, \mu\} \\
&= \sum_{o \in O_s} \mu(s,o)[\mathbb{E}\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k}|s_t = s, o_t = o\} \\
&+ \sum_{k=1}^{\infty} \sum_{s'} \gamma^k P_o(s',k) V^{\mu}(s')]
\end{aligned}$$

$$\begin{aligned}
Q^{\mu}(s,o) &= \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k})|s_t = s, o_t = o, \mu\} \\
&= \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k}|s_t = s, o_t = o\} \\
&+ \sum_{k=1}^{\infty} \sum_{s'} \gamma^k P_o(s',k) \sum_{o' \in O_{s'}} \mu(s',o') Q^{\mu}(s',o')
\end{aligned}$$

In the above formulations, the first term in both action value function and value function is the expectation on the cumulative discounted reward that the agent receives while following option o. This is a consequence of the multi-step nature of each option's execution. $P_o(s',k)$ is the probability that the agent terminates at state s' after k steps while following the policy π_o. It can be easily computed using the probabilities from the policy and the model of the environment. The above formulations can be used to solve the problem and find an optimal policy over options if a model of the environment is available. Bellman optimality operators can be defined the same way as standard Reinforcement Learning problems to find the Optimal Value and Action-value functions. In order to do so we first make some definitions.

$$r_s^o = \mathbb{E}\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} | s_t = s, o_t = o\}$$

$$p_{ss'}^o = \sum_{k=1}^{\infty} P_o(s', k)\gamma^k$$

$$V^\mu(s) = \sum_{o \in O_s} \mu(s,o)[r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')]$$

$$Q^\mu(s,o) = r_s^o + \sum_{s'} p_{ss'}^o \sum_{o' \in O_{s'}} \mu(s',o') Q^\mu(s',o')$$

r_s^o is the expected cumulative discounted reward that the agent receives while following option o starting from state s. This along with the $p_{ss'}^o$ allows us to derive simpler formulations for Value function and action-value functions. Now we can obtain the optimal values for value function and action-value function for policies over a given set of options. Methods very similar to the Dynamic Programming based algorithms for model-based Reinforcement Learning can be used to obtain the optimal value functions and therefore obtain optimal policy. In the next section, these methods are discussed. Also, for the cases where model is not available model-free sampling methods similar to TD methods can be used to obtain Optimal policies and learn value functions.

$$\begin{aligned} V_O^*(s) &= \max_\mu V^\mu(s) \\ &= \max_{o \in O_s}[r_s^o + \sum_{s'} p_{ss'}^o V_O^*(s')] \end{aligned}$$

$$\begin{aligned} Q_O^*(s,o) &= \max_\mu Q^\mu(s,o) \\ &= r_s^o + \sum_{s'} p_{ss'}^o \max_{o' \in O} Q_O^*(s',o') \end{aligned}$$

4 SMDP planning and learning

Similar to MDPs, Dynamic programming methods such as Policy Iteration and Value Iteration can be used to solve SMDPs and obtain the optimal value functions and optimal policy over options. Similar to MDPs, Bellman operators can be defined for SMDPs and they can be used to step by step improve over an approximate estimate of value functions.

$$B_\mu(V)(s) = \sum_{o \in O_s} \mu(s,o)[r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')]$$

This operator is clearly a contraction and similar to MDPs, applying this operator for multiple steps should get us closer to the actual value function for the given policy μ. Then the optimal policy for this value function can be obtained by the following. We can repeat the above step on the obtained policy to even acquire better estimates for the Value function. Repeating these two steps should get us close to the optimal policy and its value function.

$$\underset{o \in O_s}{\operatorname{argmax}}[r_s^o + \sum_{s'} p_{ss'}^o V(s')]$$

Other similar Dynamic Programming based methods such as Value Iteration can be used in this setting as well. In cases where a model of the environment is not available, similar to regular reinforcement learning problems, we can use sample based learning methods. Similar to Reinforcement Learning problems, bootstrapping from current estimates can be used to speed up learning and reduce variance. Similar to Reinforcement Learning problems some bias is expected to appear in the obtained results. The following TD update rule can be used to learn the value function for a given policy.

$$V^\mu(s) \leftarrow V^\mu(s) + \alpha[R + \gamma^k V^\mu(s') - V^\mu(s)]$$

Where R is the cumulative reward obtained from running an option o and k is the number of time-steps it takes to complete an option. s' is the state that the option finishes its execution. Similar to Reinforcement Learning problems, on-policy and off-policy methods can be used to learn a policy rather than just learn the value function of a policy. Methods such as SARSA and Q-learning are suitable for this situation as well. For example, the update rule for SARSA is given below.

$$Q(s,o) \leftarrow Q(s,o) + \alpha[R + \gamma^k Q(s',o') - Q(s,o)]$$

Where R, s' and k are similar to the TD case and o' is the option chosen after o and starting from state s'.

5 Intra Option Methods

The main problem with SMDP methods is that they are potentially very sample inefficient and therefore very computationally expensive. For example, each update in the TD method that we covered in the previous section will involve an entire option executing which makes this method significantly more inefficient than regular learning methods. Now, we cover methods where options are not treated as black boxes and information about them is known. The goal is similar to the previous section where we wanted to learn an optimal policy over options. This time, the inside information of options is used as well. Instead of waiting for the entire option is executed, we want to learn from every step of the execution. This method only applies to cases where to option policy π_o has the Markov property. We use a new notation for the value of each state option pair. Assuming that $Q^*_O(s,o)$ is the optimal state-option value function, the value of each state option pair is given below.

$$U^*_O(s,o) = (1-\beta_o(s))Q^*_O(s,o) + \beta_o(s)\max_{o' \in O} Q^*_O(s,o')$$

This equation stems from the fact that at each state s option o has a chance $\beta_o(s)$ of termination. Then, we can write $Q^*_O(s,o)$ as:

$$Q^*_O(s,o) = \sum_{a \in A_s} \pi(s,a)\mathbb{E}\{r + \gamma U^*_O(s',o)|s,a\}$$

We can easily use sampling methods to learn $Q^*_O(s,o)$. This results in an off-policy method with the following update rule.

$$Q(s,o) \leftarrow Q(s,o) + \alpha[r + \gamma U(s',o) - Q(s,o)]$$

$$U(s,o) = (1-\beta_o(s))Q(s,o) + \beta_o(s)\max_{o' \in O} Q(s,o')$$

We can still learn the optimal action-value function with significantly better sample-efficiency compared to the Semi-MDP learning methods. Therefore, this approach is highly preferable to Semi-MDP methods if the options have Markov property.

6 Learning options with subgoals

So far, we have only discussed methods for learning a policy over options. In every single method that we have discussed so far, the policy $\pi_o$corresponding to the option o has been constant. Now, we discuss an approach to learning the underlying policies for options π_o. It is natural to think that the goal of options are basically achieving sub goals which are each consequential in reaching the ultimate goal. In this approach, first we define the set $G \subset S$. This set includes the states that we want the option to terminate in. In a way, we want to encourage the learning algorithm to learn policies that follow a specific subgoal. We define another function $g : G \to$ which is an indication of how desirable it is for an option to terminate at a given state. We are basically fixing the value function for some of the states that we consider terminal or consequential in achieving a subgoal.

$$s_{t+1} \in G : Q_g(s_t, a_t) \leftarrow Q_g(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_g(s_{t+1}, a) - Q_g(s_t, a_t)]$$

$$s_{t+1} \notin G : Q_g(s_t, a_t) \leftarrow Q_g(s_t, a_t) + \alpha[r_{t+1} + \gamma g(s_{t+1}) - Q_g(s_t, a_t)]$$

Finding subgoals has received substantial focus in the recent years, as it was the best way to learn options for a long time. Many methods have been introduced that try to find subgoals for a given MDP. There have been interesting results in finding critical states that are of interest for option termination and initiation. Because of the combinatorial nature of most of these methods they can be in some cases computationally expensive and therefore not very efficient for solving difficult tasks. Overall, solving Hierarchical Reinforcement Learning problems with the methods which have been covered so far in this report does not work very well. A different approach based on the Policy Gradient theorem (R. S. Sutton, McAllester, Singh, and Mansour (2000)) has been proposed which tackles the whole problem of learning options and a policy over options in an end-to-end manner.

7 The Option-Critic Architecture

In this method which is based on the Policy Gradient family of algorithms (R. S. Sutton et al. (2000)), the policy over options is learned simultaneously with the intra-option policies and the termination condition for each option. The following Action-value function which is similar to action value functions in Intra-Option value learning is used.

$$Q_O(s, o) = \sum_{a \in A_s} \pi(s, a)(r + \gamma \sum_{s'} P(s'|s, a)U(s', o))$$

$$U(s, o) = (1 - \beta_o(s))Q_o(s, o) + \beta_o(s)V_O(s)$$

Both option policies and option termination functions are differentiable and therefore gradients for them can be computed. In the diagram in 1 the overall procedure for the Option-Critic method is shown. Similar to Actor-Critic methods, a neural network called the Critic approximated the action value functions and that is later used in parameter update as a weight for the gradient of Logarithms of the probability of action while following a policy. Due to the discrete nature of the actions by the policy over gradients, tabular methods can be used for learning the policy over options. The gradients of the action-value function with respect to option parameters are included in Bacon et al. (2016). θ_o are the parameters for policy of option o and ν_o are the parameters for the termination condition of option o. The update rules for θ_o and ν_o are:

$$\theta_o \leftarrow \theta_o + \alpha_\theta \frac{\partial \log \pi_{o,\theta}(a|s)}{\partial \theta} Q_U(s, o, a)$$

$$\nu_o \leftarrow \nu_o - \alpha_\nu \frac{\partial \beta_{o,\nu}(s')}{\partial \theta} (Q_O(s', o) - V_O(s'))$$

In the above equation, $Q_U(s, o, a)$ is an approximation for $r + \gamma \sum_{s'} P(s'|s, a)U(s', o)$.

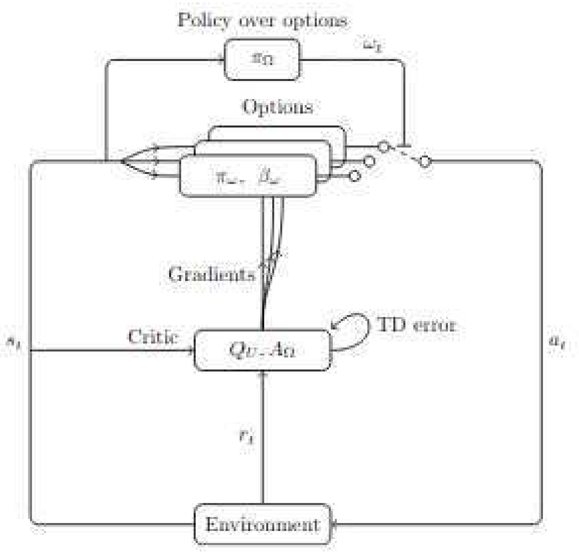

Figure 1: A diagram of the option-critic architecture(figure taken from Bacon et al. (2016))

8 Discussion and Conclusions

Options is one of the most promising approaches to solving the problem of Temporal Abstraction in Hierarchical Reinforcement Learning. Before the introduction of the Option-Critic method, learning policies over options or learning options themselves were usually separated problems and most methods tried to learn one of the two while keeping the other fixed. With the Option-Critic method though both can be learned in an End-to-end manner which makes the problem of learning options at a larger scale feasible and this is the main contribution of Bacon et al. (2016). The main limitation of the Option-Critic method is its inability to learn initiation sets for options and therefore allowing all options to be executed at every state of the environment.

References

Bacon, P.-L., Harb, J., & Precup, D. (2016). *The option-critic architecture.*

Dietterich, T. G. (1999). *Hierarchical reinforcement learning with the maxq value function decomposition.*

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181-211.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, & K. Müller (Eds.), *Advances in neural information processing systems 12* (pp. 1057–1063). MIT Press. Retrieved from `http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf`

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). *Feudal networks for hierarchical reinforcement learning.*