Hierarchical RL

July 19, 2017

1 Option MDP

A Markovian option $\omega \in \Omega$ is a triple $(\mathcal{I}_\omega, \pi_\omega, \beta_\omega)$, where $\mathcal{I}_\omega \subset \mathcal{S}$ is an initiation set, π_ω is an intra-option policy, and $\beta_\omega : \mathcal{S} \mapsto [0, 1]$ is a termination policy.

For the higher level, $V_\Omega(s)$ is the value function over options and $Q_\Omega(s, \omega)$ is the option-value function, with the following relation

$$V_\Omega(s) = \sum_\omega \pi_\Omega(\omega \mid s) Q_\Omega(s, \omega)$$

For the lower level, $U : \Omega \times \mathcal{S} \mapsto \mathbb{R}$ is the option-value function upon arrival, where $U(\omega, s)$ measures the value of executing ω upon entering a state s, which is defined as

$$U(\omega, s) = \underbrace{(1 - \beta_\omega(s)) Q_\Omega(s, w)}_{\text{continue with } \omega} + \underbrace{\beta_\omega(s) V_\Omega(s)}_{\text{terminate } \omega}$$

Further, $Q_U(s, \omega, a) : \mathcal{S} \times \Omega \times \mathcal{A} \mapsto \mathbb{R}$ is the action-value function, which is defined as

$$Q_U(s, \omega, a) = r(s, a) + \gamma \sum_{s'} P(s' \mid s, a) U(\omega, s')$$

The transition of the Markov chain can be written as

$$P(s_{t+1}, \omega_{t+1} \mid s_t, \omega_t) = \sum_a \pi_{\omega_t}(a \mid s_t) P(s_{t+1} \mid s_t, a) \left[\underbrace{(1 - \beta_{\omega_t}(s_{t+1})) \mathbf{1}_{w_t = w_{t+1}}}_{\text{continue with } \omega_t} + \underbrace{\beta_{\omega_t}(s_{t+1}) \pi_\Omega(w_{t+1} \mid s_{t+1})}_{\text{terminate } \omega_t \text{ and choose } \omega_{t+1}} \right]$$

or staring from state (s_t, ω_{t-1}), the transition is

$$P(s_{t+1}, \omega_t \mid s_t, \omega_{t-1}) = \left[(1 - \beta_{\omega_t}(s_t)) \mathbf{1}_{w_{t-1} = w_t} + \beta_{\omega_{t-1}}(s_t) \pi_\Omega(w_t \mid s_t) \right] \sum_a \pi_{\omega_t}(a \mid s_t) \gamma P(s_{t+1} \mid s_t, a)$$

2 Notes on recent papers

Paper list

- Unified Inter and Intra Options Learning Using Policy Gradient Methods

- Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation
- A Deep Hierarchical Approach to Lifelong Learning in Minecraft
- Learning and transfer of modulated locomotor controllers
- The Option-Critic Architecture
- Adaptive Skills Adaptive Partitions (ASAP)
- Stochastic Neural Networks for Hierarchical Reinforcement Learning
- Strategic Attentive Writer for Learning Macro-Actions
- FeUdal Networks for Hierarchical Reinforcement Learning
- A Laplacian Framework for Option Discovery in Reinforcement Learning
- Deep Reinforcement Learning with Macro-Actions
- Multi-Level Discovery of Deep Options (Imitation learning)
- Probabilistic Inference for Determining Options in Reinforcement Learning (Probabilistic Reinforcement Learning)
- Options Discovery with Budgeted Reinforcement Learning
- Deep Successor Reinforcement Learning

The core motivation of hierarchical reinforcement learning is to shorten the time scales (reward delay) the overall policy will face, and thus make the learning problem easier. Whether the learned hierarchy can generalize well to unseen cases is an additional problem not necessarily of importance.

A core fundamental problem in hierarchical reinforcement learning is what the hierarchy is like or where the hierarchy comes from:

- Using human knowledge, it is easy to design a hierarchy for specific learning problems. With the hierarchy specified, one can directly tackle the entire big problem, or take advantage of curriculum learning and incremental learning to solve the problem bottom up.
- Another possibility is to discover / learn the hierarchy from the data together with the policy. Firstly, there can be identifiability issue here since there can be many possible hierarchies with the same performance. Secondly, as far as I know, there is no good measure to evaluate the goodness of a learned hierarchy.

Another problem is how to train the policies on different levels. In other words, what is the target of policies on each level? This choice essentially decides the time scale each level deals with, and thus has a crucial influence on the difficulty of learning.

2.1 Unified Inter and Intra Options Learning Using Policy Gradient Methods

Augmented state $\tilde{s} = (i, s)$, where $i \in \mathcal{O}$ is the the option with which we arrive at the state s.

Augmented action $\tilde{a} = (\phi, j, a)$ where $\phi \in A_{\text{stop}}$ is the decision whether to stop the current *arrive*-option i. The action

$j \in \mathcal{O}$ is the choice of the *act*-option, and $a \in A$ is the primitive action chosen by the act-option j.

Given the original transition probabilities $P(s' \mid s, a)$ we can calculate the transition probability function in the augmented one:

$$P(\tilde{s}' \mid \tilde{s}, \tilde{a} = P\left((i', s') \mid (i, s), (\phi, j, a)\right) = \mathbf{1}_{\{i'=j\}} P(s' \mid s, a)$$

Then, we define the following policy Π in the augmented space:

$$\Pi\left((\phi, j, a) \mid (i, s)\right) = P_{sp}(\phi \mid i, s) P_{op}(j \mid \phi, i, s) P(a \mid j, s),$$

where

$$\begin{aligned} P_{sp}(\phi \mid i, s) &= \mathbf{1}_{\{\phi=\text{stop}\}} \beta_i(s) + \mathbf{1}_{\{\phi=\text{cont}\}} (1 - \beta_i(s)) \\ P_{op}(j \mid \phi, i, s) &= \mathbf{1}_{\{\phi=\text{stop}\}} \mu(j \mid s, i) + \mathbf{1}_{\{\phi=\text{cont}\}} \mathbf{1}_{\{j=i\}} \\ P(a \mid j, s) &= \pi(a \mid s) \end{aligned}$$

It can be proved that both MDPs induce the same probability measure over augmented state-actions trajectories (which include stopping decisions and option choices), $\{i_t, s_t, \phi_t, j_t, a_t\}_{t=0}^{T}$ particularly:

$$E^{\{\mu, \mathcal{O}\}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_o, i_o \right] = E^{\Pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid (i_o, s_o) \right]$$

Suppose at time t the process arrives a state st with a leaf option o_{i_t, n_t}:

1. Decide whether to stop the current leaf option.
2. The hierarchy above, μ_{i_t} , makes a decision whether to stop the execution of the current level 1 decision node.
3. The root policy μ_0 chooses a new level 1 decision node, μ_{j_t}.
4. The level 1 hierarchy, μ_{j_t} chooses a new leaf option o_{j_t, m_t}
5. The new leaf option chooses a primitive action $a_t \sim \pi_{j_t, m_t}(\cdot \mid s_t)$

2.2 Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation

Model:

- controller: $\mathcal{S} \times \mathcal{O} \mapsto \mathcal{A}$, trained by intrinsic reward $r(g)$ generated by an oracle, i.e., $R_t(g) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g)$
- meta-controller: $\mathcal{S} \mapsto \mathcal{O}$, trained by extrinsic reward f_t, i.e., $F_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} f_{t'}$. Note that f_t aggregates several rewards obtained from controller execution.

Details:

- A goal is a binary mask of the same size as the perceptual image
- Deep Q-learning is used for training
- Pretrain the controller to solve an arbitrary task indicated by an arbitrary mask

Comments:

- Additional information is given in the form of intrinsic reward

2.3 A Deep Hierarchical Approach to Lifelong Learning in Minecraft

Model:

- controller: $\mathcal{S} \mapsto \mathcal{A}$, pretrained on simpler task by extrinsic reward
- meta-controller: $\mathcal{S} \mapsto \mathcal{O} \cup \mathcal{A}$, trained on complex task by extrinsic reward. When an option is selected, the controller execute a sequence of actions until it terminates. All the rewards in the time interval are aggregated as a single reward to the meta-controller

Details:

- Policy distilling is used to learning a single lower-level controller
- Deep Q-learning is used for training
- Not clear how the termination works

2.4 Learning and transfer of modulated locomotor controllers

Model:

- controller: $\mathcal{X} \times \mathcal{O} \mapsto \mathcal{A}$
- meta-controller: $\mathcal{S} \mapsto \mathcal{O}$ which outputs a dense vector representing the option/goal

Details:

- $\mathcal{X}$ is a subset of $\mathcal{S}$ which is task independent.
- Meta-controller omits an option every K steps
- Reparameterization trick is used for the meta-controller, making the entire model differentiable. Thus, the entire model is trained by policy gradient (A3C) from the action probability
- There is a pretraining stage where the task is simpler. After pretraining, the meta-controller is discard and the controller is retained.

2.5 The Option-Critic Architecture

Model:

- action (intra-option) policy $\pi_A : \mathcal{S} \mapsto \mathcal{A}$
- option (inter-option) policy $\pi_O : \mathcal{S} \mapsto \mathcal{O}$ which is a multinomial policy over all possible options

- termination policy $\beta : \mathcal{S} \mapsto \{0, 1\}$

Objective:

$$\max \rho(\pi_O, \pi_A, \beta) = \mathbb{E}_{\pi_O, \pi_A, \beta, P} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$$

where P is the transition probability of the MDP.

The entire thing is trained by policy gradient from the action probability

2.6 Adaptive Skills Adaptive Partitions (ASAP)

The proposed formulation is very similar to the option-critic architecture, except that the proof is based on episodic case and looks simpler. In comparison, the proof in the option-critic architecture paper considers the infinite horizon case and relies on unrolling the Bellman equation to complete the proof (similar to the proof of the policy gradient theorem).

Detailed similarities are as follows

- The critic approximates the option-action-state value $Q(s, o, a)$ in both paper.
- The option-action-state critic signal at **every** step is used to train all model parameters end-to-end. As a result, the higher-level option policy (almost) deals with the same time scale as the lower-level action policy.

Despite the similarity, there are still many detailed differences between these two papers

- The ASAP paper used combinatorial representation for options.
- The ASAP paper is essentially an episodic policy gradient algorithm.
- The option policy is trained by PG in ASAP while by intra-option Q-learning in option critic.

2.7 Stochastic Neural Networks for Hierarchical Reinforcement Learning

$o \sim \mathrm{Cat}(\frac{1}{K})$ corresponds to a specific option. The action policy associated with the option is produced by a neural net, i.e., $\pi_A(a \mid x, o) = F(x, o)$.

The option policy outputs a categorical distribution $\pi_O(o \mid s)$, sample a one-hot option o, and commits to the option for K steps.

Details:

- Use mutual information maximization to encourage different option codes really correspond to different behaviors.
- Low-level intra-option policies are frozen after pretraining.
- The reward in the pretraining stage is task-agnostic, such as the moving speed in a plain environment.

2.8 Strategic Attentive Writer for Learning Macro-Actions

- The agent maintains a multi-step action plan, periodically updates the plan based on observations and commits to the plan between the replanning decision points.
- The replanning decisions as well as the commonly occurring sequences of actions, i.e. macro-actions, are learned from rewards.

Action-plan: $\mathbf{A}^t \in \mathbb{R}^{|A| \times T}$

Commitment plan: $\mathbf{c}^t \in \mathbb{R}^{1 \times T}$

Attentive read and write are used for plan updates:

- $\mathbf{D} = write(\mathbf{p}, \psi_t^A)$ where $\mathbf{p} \in \mathbb{R}^{A \times K}$ and $\mathbf{D} \in \mathbb{R}^{A \times T}$
- $\beta = read(\mathbf{A}^t, \psi_t^A)$ where $\beta_t \in \mathbb{R}^{A \times K}$

Additional details:

- Inject noise into the state using a noisy channel with re-parameterization trick. This is used as a structured exploration technique.
- Straight-through estimator is used for the binary update decision sample from update probability.

Comments:

- In terms of hierarchical reinforcement learning, the biggest distinction of this paper is that it enables the concept of macro-action in a dynamic and implicit way. Thus, such macro-actions (options) are discovered automatically without using any prior knowledge.
- In addition, the macro-action has a flavor of planning. In a deterministic environment, it seems to be a reasonable approach. The character prediction on Penn treebank is actually very interesting, showing us a trade-off between long time scale (character-level) and large option space (word-level). This problem arises probably because of the initialization set of each option (word) is all the state.
- The fact that the macro-action is dynamic could also be a disadvantage of this method. Basically, with some noise injected to the hidden state that generates the action-plan, the precision of the macro-action cannot be guaranteed. Therefore, the generalization might not be good.

2.9 FeUdal Networks for Hierarchical Reinforcement Learning

Motivation: when options are learned end-to-end, they tend to degenerate to one of two trivial solutions:

(i) only one active option that solves the whole task;

(ii) a policy-over-options that changes options at every step, micro-managing the behavior.

Core idea: higher level produces a meaningful and *explicit* goal for the lower level to achieve. Specifically, sub-goals emerge as directions in the latent state space.

The higher level outputs a goal vector g_t at each time step, and it is trained by the following gradient

$$\nabla_\theta g_t(\theta) = A_t \nabla_\theta \text{cosine}(s_{t+c} - s_t, g_t(\theta))$$

which encourages $g_t(\theta)$ to be similar to $s_{t+c} - s_t$ in terms of cosine distance if the advantage $A_t = R_t - V_t(x_t)$ is positive and vice versa. Thus, a well trained g_t has a semantic meaning as an advantageous direction in the latent state space at a horizon c, which defines its temporal resolution.

The intrinsic reward to lower level is

$$r_t^I = \frac{1}{c} \sum_{i=1}^{c} \text{cosine}(s_t - s_{t-1}, g_{t-i})$$

which encourages the lower-level to take actions to change the state such that the goals g_{t-i} can be achieved.

Transition policy gradient is the mostly interesting part of the paper, which essentially shows the followings:

- If we assume the transition in latent space with horizon c, $s_{t+c} - s_t$, follows a von Mises-Fisher distribution, and the goal vector g_t is the predicted mean of the distribution, then we have

  $$p(s_{t+c} - s_t \mid g_t) \propto e^{\text{cosine}(s_{t+c} - s_t, g_t)},$$

  or equivalently

  $$p(s_{t+c} \mid s_t, g_t) \propto e^{\text{cosine}(s_{t+c} - s_t, g_t)}.$$

  Based on this assumption, the heuristic update rule of the higher level can be obtained by applying the policy gradient theorem. Surely, the assumption is most likely to be wrong, and thus the training objective for the higher level does not follow the direction of the true semi-Markov policy gradient.

- More over, the intrinsic reward $r_t^I = \frac{1}{c} \sum_{i=1}^{c} \text{cosine}(s_t - s_{t-1}, g_{t-i}) = \frac{1}{c} \sum_{i=1}^{c} \log p(s_t \mid s_{t-i}, g_{t-i}))$ is based on the log-likelihood the action trajectory. In other words, given a goal g_t specified by the higher level controller, the intrinsic reward encourages the lower level to generate actions (and thus states) that maximize the log-likelihood.

To summarize, the higher level decides where to go in the latent space in order to improved expected return while the lower level is concerned with how to reach the goal (in the latent space) set by the higher level.

2.10 A Laplacian Framework for Option Discovery in Reinforcement Learning

Using graph Laplacian to discover option without extrinsic reward.

The learned representation informs the agent what options are meaningful to be sought after.

2.11 Multi-Level Discovery of Deep Options

Imitation learning with latent variable.

2.12 Probabilistic Inference for Determining Options in Reinforcement Learning

Define the target distribution $p(s,a)$ as advantage re-weighted distribution of $q(s,a)$ by which the trajectory is generated on

$$p(s,a) \propto q(s,a) \exp\left(\frac{Q(s,a)-V(s)}{\eta}\right)$$

To push the current policy to the target policy, we can minimize the KL divergence as follows

$$\begin{aligned}
\pi^*(a \mid s) &= \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}_{p(s)}\left[\mathrm{KL}(p(a \mid s)||\pi(a \mid s)\right] \\
&= \underset{\pi}{\operatorname{argmin}} \int p(s,a) \log \pi(a \mid s) dsda + \text{const} \\
&= \underset{\pi}{\operatorname{argmin}} \int q(s,a) \exp\left(\frac{Q(s,a)-V(s)}{\eta}\right) \log \pi(a \mid s) dsda \\
&= \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}_{q(s)} \mathbb{E}_{a\sim q(a|s)} \exp\left(\frac{Q(s,a)-V(s)}{\eta}\right) \log \pi(a \mid s)
\end{aligned}$$

2.13 Deep Reinforcement Learning with Macro-Actions

Use heuristics-based macro-actions in Q-learning.

2.14 Options Discovery with Budgeted Reinforcement Learning

Model:

- Acquisition Model: $P(\sigma_t = 1) = \mathrm{sigmoid}(f_{acq}(h_{t-1}, x_t))$ deciding whether a new high-level observation y_t needs to be acquired
- Option Model: $o_t = \begin{cases} o_{t-1}, & \sigma_t = 0 \\ \mathrm{GRU}_{opt}(x_t, y_t, o_{t-1}), & \sigma_t = 1 \end{cases}$
- Action Model: $h_t = \begin{cases} \mathrm{GRU}_{opt}(x_t, h_{t-1}), & \sigma_t = 0 \\ o_t, & \sigma_t = 1 \end{cases}$ and $\pi(a_t \mid s_t) = \mathrm{softmax}(f_{act}(h_t))$

Budgeted learning: $\tilde{r}(s_t, a_t, \sigma_t) = r(s_t, a_t) - \lambda\sigma_t$

3 Random Thoughts

Why should *automatic option discovery* improve the performance of an Agent? Since there is no free lunch, in order to achieve any improvement, there must be some additional knowledge / prior injected into the procedure.

Firstly, the some options defined, we essentially create a new augmented MDP with an additional transition arch. However, note that the optimal trajectory should remain the same since the introduction of the option does not change the environment. As a result, for a given task, there are two types of options:

(1) the option is a sub-sequence of the optimal trajectory

(2) the option is not a sub-sequence of the optimal trajectory, which means that using the option is not the optimal behavior.

A natural thinking will regard the first type of option more desirable than the second type. While this is true at the end of training, this is not necessarily true during training or more specifically during exploration. During exploration, the really interesting quantity is the minimum number of decisions an agent needs to make that can take the agent to the target (rewarding) state. So, the key problem is whether an option is one of the shortest steps that takes the agent to the goal.

4 What makes a good option

- states or regions that the agent tends to visit frequently
- properties of the graph defined by the MDP's transition probabilities were used: betweenness, border states of strongly connected regions
- clustering of the state space
- Bayesian model selection

References