Hierarchy Deep Q-Learning Framework

October 4, 2016

At first, we construct a hierarchy framework without deep neural network. After we specify all tasks, the set containing all tasks is defined as $T : \{T_1, T_2, T_3...\}$. We also define the available action set as $A : \{a_1, a_2, a_3...\}$. We define $O : T \cup A$, which is often used as all the operation we can perform. Besides, we define a global task named *global*, which only focus on getting more extrinsic reward during the game. We define $G : T \cup \{global\}$, which is often used as the super-task we are going through. So a task can be regarded as an operation as well as a goal. Ω means the state set.

Then, we define tasks as follows(containing actions as primitive tasks):

$$O : \{ \quad \pi : \Omega * O \to [0, 1], \\ \beta : \Omega \to [0, 1] \quad \} \tag{1}$$

The π specifies the policy: $\pi_o(s, t)$ mean the policy of o: the probability of taking task t under the state s. and β defines the termination condition of a task, $\beta_o(s)$ denotes the probability of terminating the task at the state of s, meaning the goal of it. One thing notable is that primitive actions are also compatible for the subtask definition, while the policy $\pi(s, o) = 1$ for all states only when o is the current action and $\beta(s) = 1$ for all states, meaning the task will definitely stop after one action.

Now we can define the learning procedure.

1 Hierarchy Model Learning

We have to define different intrinsic reward based on the task on which the subtask is performed.

$r_s^a \quad a \in A, s \in \Omega :$ extrinsic reward gained when perform primitive action a on the state of s, it's given by the game model.

Because actions can be regarded as primitive tasks, we can define a more general reward:

$r_s^o \quad o \in O, s \in \Omega :$ extrinsic reward gained when perform task o on the state of s.

Then we define the mixture reward under the hierarchy reinforcement learning:

$r^o_{s,g} \quad o \in O, s \in \Omega, g \in G$: mixture reward gained when perform subtask o under the task goal of g on the state of s.

Then we define the Bellman equation of reward and the intrinsic part of reward:

$$r^o_s = \sum_{t \in O} \pi_o(s,t) E[r^t_s + \gamma^k (1 - \beta_o(s')) r^o_{s'})] \tag{2}$$

In the equation, k is the number of steps used performing t from state s. The reward for task o under state s is the expectation of the sum of the accumulate reward gained during the process of the first subtask t and the future expected reward if the task is not terminated according to $\beta_o(s')$. Primitive action also satisfies this equation, but will be given directly from the model.

$$r^o_{s,g} = \textstyle\sum_{t \in O} \pi_o(s,t) E[IR(g,s') + r^t_s + \gamma^k (1 - \beta_o(s')) r^o_{s'}] \tag{3}$$

$$IR(g,s) = \rho * \beta_g(s) \tag{4}$$

We define the mixture reward with two parts: getting more rewards while trying to reach specified states. Parameter ρ is used to control the relative weight of the local goal of task g. Under the reward model, we can construct the value learning procedure. One exception is that when $g == global$, $IR(g,s) = 0$, we only consider maximum extrinsic reward.

This procedure helps building the model of the current hierarchy structure.

2 Hierarchy Value Learning

In the value learning procedure of hierarchy tasks. We define a new Q-function. $Q^*(g,s,o) \quad g \in G, s \in \Omega, o \in O$ denotes the max Q-value after performing task o from the state s under the goal of task g, if from that time on we always conduct the optimal operations.

We get the Bellman equation of this new Q-value as follows:

$$Q^*(g,s,o) = \sum_{t \in O} \pi_o(s,t) E[r^t_{(s,g)} + \gamma^k ((1 - \beta_o(s')) Q^*(g,s',o) + \beta_o(s') \max_{t' \in O} Q^*(g,s',t'))] \tag{5}$$

k is the number of primitive steps used for task t to finish from state s. We notice that this way of learning can be done just after every finishing of subtask of o. It also satisfied when $g == global$.

This procedure get the Q-value from the current hierarchy structure, which will also dynamically change the policy of tasks to building better hierarchy structure.

3 β learning

The goal of random initialized tasks should also be learned to adapt the goal to maximum Q-value in section 2. We use interrupting policy to help learning the

Algorithm 1 Deep Hierarchy Reinforcement Learning Procedure

1: Update the reward with Bellman equation of reward (equation 2).
$r^o_s \leftarrow r^o_s + \alpha[r^t_s + \gamma^k(1 - \beta_o(s'))r^o_{s'}) - r^o_s]$

2: Update the Q-value with Bellman equation of Q-value.
$Q^*(g, s, o) \leftarrow Q^*(g, s, o) + \alpha[r^t_{(s,g)} + \gamma^k U(g, s', o) - Q^*(g, s, o)]$
where,
$U(g, s, o) = (1 - \beta_o(s))Q^*(g, s, o) + \beta_o(s) \max_{t' \in O} Q^*(g, s, t')$

3: Consider if the subgoal of task o is finished by $\beta_o(s)$.

4: Consider whether interrupting, if so, update the $\beta_o(s)$ as described in Section 3.

β function as well as choose better policy.

In the running phase, when $Q^*(g, s, currento) < \max_{t \in O} Q^*(g, s, t)$, then interrupt the task *currento* to choose task $\underset{t \in O}{argmax}\, Q^*(g, s, t)$ with probability $sigmoid(\lambda * (\max_{t' \in O} Q^*(g, s', t') - Q^*(g, s', o)))$. λ controls the balance between keeping the current policy or changing to better policy.[2] has proved this policy will definitely improve (at least not deteriorate) V-value. Meanwhile, the β of is updated. The update way is inspired by the new Q-function. The first derivative of $Q^*(g, s, o)$ with respect to $\beta_o(s')$ from equation 5 is as follows:

$$\frac{\partial Q^*(g, s, o)}{\partial \beta_o(s')} = \sum_{t \in O} \pi_o(s, t) E[\gamma^k (\max_{t' \in O} Q^*(g, s', t') - Q^*(g, s', o))] \tag{6}$$

To maximum Q value we take use of the equation. When a subtask t is finished, we accumulate the grad value $grad' = grad + (\max_{t' \in O} Q^*(g, s', t') - Q^*(g, s', o)))$, Then:

$$\beta_o(s) = sigmoid(grad) \tag{7}$$

This procedure helps building better hierarchy structure by changing the goal of tasks.

4 Deep Learning Procedure

Predefine the number of tasks as n. We may need three DNNs: one for reward mapping $r(s, o)$, one for Q-value mapping $Q^*(g, s, o)$ and one for $\beta(s, o)$. To calculate $\max_{t' \in O} Q^*(g, s', t')$ in one run of NN, the o parameter will be fixed in the output of the NN as in [1]. To make g as a input of NN of Q-value, we will assign every task a distributed vector as the input. The structure of NN has not decided.

During the reinforcement learning, the model will keep finding the next task to perform based on the $\pi_o(s, t)$, we use a probabilistic definition of it.

$$\pi_o(s, t) = \frac{exp(\mu * Q^*(o, s, t))}{\sum_{t' \in o} exp(\mu * Q^*(o, s, t'))} \tag{8}$$

The parameter μ is used to control the balance between the max-Q policy and softmax-Q policy.

When one subtask t is finished from the state s to s' under the super-task and current-task (g, o), learning procedure begins as in Algorithm 1.

There are 3 new parameters in learning procedure : ρ, λ, μ, their use are explained in the text. ρ should decrease during the learning procedure, gradually making learning focus on extrinsic reward rather than intrinsic reward. λ should gradually increase, because when Q-value is sufficiently learned, the interrupting policy will be more credible. μ should gradually increase, for the same reason for λ, task policy should tend to be more deterministic to max-Q policy.

References

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

[2] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.