Skill Combination for Reinforcement Learning

Zhihui Luo, David Bell, Barry McCollum

School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast
{zluo02, da.bell, b.mccollum }@qub.ac.uk

Abstract. Recently researchers have introduced methods to develop reusable knowledge in reinforcement learning (RL). In this paper, we define simple principles to combine skills in reinforcement learning. We present a skill combination method that uses trained skills to solve different tasks in a RL domain. Through this combination method, composite skills can be used to express tasks at a high level and they can also be re-used with different tasks in the context of the same problem domains. The method generates an abstract task representation based upon normal reinforcement learning which decreases the information coupling of states thus improving an agent's learning. The experimental results demonstrate that the skills combination method can effectively reduce the learning space, and so accelerate the learning speed of the RL agent. We also show in the examples that different tasks can be solved by combining simple reusable skills.

1 Introduction

Reinforcement learning offers a fundamental framework for intelligent agents to improve their behavior through interacting with the environment. In a delayed feedback environment, RL proves to be a feasible way to train an agent to perform at a high standard. Recently within the RL domain, there has been increased interest in transferable learning which attempts to reuse learned control knowledge across different problems. In recent research, one outlined method is to build portable knowledge in the option framework using agent space [1]. Another method uses rule transfer to assist learning that target tasks from source tasks [2]. These methods show promising results. However, they do not provide methods to represent tasks on a higher level in reinforcement learning. In this paper, we address the problem of how to represent and combine learned skills to make it suitable for different tasks. Thus an agent can learn quickly. Moreover, this assists the agent to construct a suitable and more compact representation for tasks, which guides the agent's learning process.

2 Skills in Reinforcement Learning

In this section we introduce the problem domain. In the context of reinforcement learning, a problem domain is an environment which consists of more than one sub-

problem, and each of the sub-problems can be learned by an agent independently. An agent attempts to solve a sub-problem as a task in the problem domain.

Given an environment modeled as MDP - Markov Decision Process [3], a problem domain can be denoted as $M = (S, A, P, R)$. State space S represents a finite set of states in the environment. A set of actions that can be executed by the agent is denoted as A. R is denoted as the reinforcement values receive from environment, and P is representing the state transition probability.

Environmental features are properties that can be observed and measured by the agent. In this paper, we concentrate on those environment features which can be independently perceived and interacted with by the learning agent. Hence, the agent can learn to interact with each of the features separately to form skills. Suppose there are k numbers of features in the environment. Each of these features indicates an object in the environment. When the agent perceives the ith object, the agent naturally forms a state space representing the condition of this specific object in the environment. The state space of the object is denoted as S^i. All these k number of objects form an MDP's state space S, so we have:

$$S = \left\{S^1, S^2, S^3, \cdots S^k\right\} = \left\{S^i \mid i = 1, ..., k\right\}$$

From this definition, we know S^i is a sub-state space of S, so $S^i \subset S$. Each of these sub-state spaces represents an independent feature of the environment.

If an agent can learn each S^i's value separately without being affected by or interfering with other sub-states spaces, we call this learning process skill training. Formally, a skill is an ability of the agent, learned or acquired through training, to perform actions to achieve a desired goal within a problem domain. So the agent can be trained to develop a skill to execute optimal actions in a sub-state space S^i.

$$Skill^i : S^i \mapsto A^i$$

A skill of an agent can be provided by a supervised instructor or learned by unsupervised learning [4]. In this paper, our experimentations are in the reinforcement learning context, so all skills are trained using RL algorithms. In the next section we present our methods to use the combination of these skills to solve more complex tasks within a problem domain.

3 Skills Combination

In this section, we will define three basic principles to combine skills. These principles provide fundamental operators between two skills on a higher level of representation, which are independent from low level skill learning.

Sequential Combination: If skill g finishes then consequently skill h begins. We denote the sequential combination as:

$$C_{seq} = g \rightarrow h \tag{1}$$

The sequential operator $\rightarrow$ is weakly constrained between two skills. The requirements for it are the following properties:

1. Termination exists - if β is the termination condition of skill *g* and S^g is the state space of skill *g*, then there at least exists one terminal state s^g satisfying condition β : $\exists s^g \subset S^g$, *where* β *is satisfied*

2. Initialization exists - similarly, if ω is the initialization condition of skill *h*, and S^h is the state space of skill *h*, then there at least exists one initialization state s^h satisfying ω : $\exists s^h \subset S^h$, *where* ω *is satisfied*

3. State inheritable - if terminal state s^g is a sub-state of problem domain's state *s*, then initialization state s^h is also a sub-state of *s*. This property ensures termination state s^g can be inherited by initialization state s^h of the consequences skill: $(s^g \subset s) \Rightarrow (s^h \subset s)$

4. A sequential combination links one skill after another. When skill *g* finishes, the condition of the agent will pass to subsequence skill *h*. Consequently the agent's action is determined by the subsequence skill, the features relating to skill *g* can be ignored.

Concurrent Combination: Skill *g* and skill *h* will be performed by the agent concurrently. We denote concurrent combination as:

$$C_{con} = g \cap h \tag{2}$$

The concurrent operator $\cap$ is a tight binding between the skills, and satisfies the following properties:

1. Combined action – if skill *g* generates action *a* and skill *h* generates action *b* at the same time, then the combined action is denote as: $\overline{ab}$

2. State compatible- if s' is the successor state after a combined action $\overline{ab}$, then the successor state s' must be a state in both state spaces S^h and S^g. This property makes sure that when two skills are executed at the same time by the agent, they won't generate a new state that does not exist in both of the skills. $\forall s' \subset (S^h, S^g)$

3. When skills are concurrently combined, the agent's action is determined by the combined action $\overline{ab}$.

Optional Combination: Nondeterministic branch - skill *g* or skill *h* to be selected. We denote optional combination as:

$$C_{opt} = g \cup h \tag{3}$$

The optional operator $\cup$ is a weak binding of skills. It has the following properties:

1. Initialization exists - if β, ω are the initialization conditions of skill *g* and *h*, and S^g, S^h are the state spaces of skill *g* and *h*, then at least there exists one initialization state *s* satisfies β and ω at the same time: $\exists s \subset S^g$, *where* β *is satisfied* and $\exists s \subset S^h$, *where* ω *is satisfied*

2. Comparable selection – if β, ω are the initialization conditions of skills *g* and *h*, then β and ω can be compared: $\beta \geq \alpha$ or $\alpha \geq \beta$

3. After comparing the initialization conditions of skills, the agent will select and perform one of the skills. Hence the agent's action is determined by the selected skill. The features relating to the other skills can be ignored.

A common example of using skill combination is in the car driving domain. Suppose a driver learns the following skills: turnRight, turnLeft, goFoward, slowDown and signalLight. Then the driver can combine these skills to solve different tasks. For example, before a crossroads, a car driver needs to determine which way to go. This task can be represents as:

$$\text{slowDown} \cap (\text{turnRight} \cup \text{turnLeft} \cup \text{goFoward})$$

If the driver decides to go right, the composite task would be:

$$\text{signalLight} \rightarrow \text{turnRight}$$

If all the driving skills are well trained, the driver can use them to solve complex tasks on the road. We can see from the examples that by using skill combination, a more flexible knowledge representation can be used to represent different tasks in standard reinforcement learning method, and skills can be reused on different tasks.

4 Experiments

4.1 Cat and Mouse Domain Introduction

In the cat and mouse domain, we have a reinforcement learning agent - Mouse (A). We also have predator - Cat (C), food - Cheese (Z) and Home (H). The cat and mouse can each move one step forward in any direction – vertically, horizontally or diagonally. They can also choose to stay still. So there are nine possible movements. From the point of view of the mouse agent, objects in its environment - cat, cheese and home - are features of the domain.

In order to learn skills, the agent has sensors to detect the direction and distance from every feature in the environment. In this experimental setting, the agent has sensors to perceive the cat, cheese, and home in the environment. The agent can detect an object in 8 directions in the grid world (See Fig. 1). The object gives signal sharing its distance to the agent. Using the direction and distance data from the sensor, the agent can form the state space of a basic skill related to that feature.

4.2 Skill Training

Skills are independent, low-level learning problems in the environment, so they can be acquired before being applied to tasks. We design three types of skills: skill to get cheese, skill to keep away from cat and skill to store cheese. In the experiments, all skills are trained in a 10*10 grid environment using the Q-learning algorithm [5, 6] with the following standard learning parameters: ε-greedy policy, alpha (α) =0.1, gamma (γ) =0.9, epsilon (ε) =0.2. To facilitate our subsequence experiments with more comparative results, all these skills are trained to different degrees. Each of the skills is saved on the following training episodes: 200, 400, 600, and 800. So in our task learning experiments below, we can equip the agent with skills that have

different levels of proficiency. All experiences of trained skills will be saved for further use on the subsequence tasks.

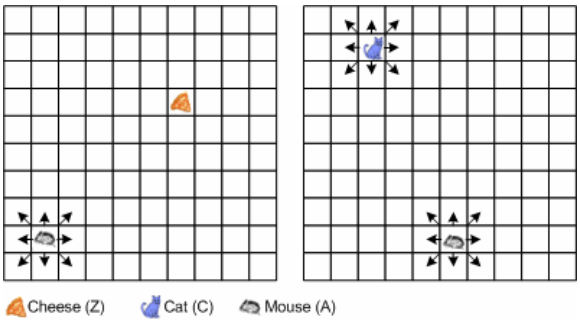

Fig. 1. Left: a skill training environment for getting cheese. Right: a skill training environment for keeping away from cat.

4.3 Task 1: Get Cheese

In this section, we demonstrate how to use skills in a task. In task 1 (See figure 2 left), the mouse agent target is to complete a 2-skill task to get a piece of cheese in a 20*20 grid environment, in the mean time the agent should try its best to avoid being caught by an approaching cat. To finish this task, the agent needs to combine the concurrent skills of getting the cheese and keeping away from the cat. Following the definition of skill's concurrent combination above (2), this task can be expressed as:

$$GC(Z) \cap KA(C)$$

$GC(Z)$ represents a skill of getting the cheese and $KA(C)$ represents a skill of keeping away from the cat. They are concurrently performed by the agent, so the concurrent operator $\cap$ is used to express the relationship between the two skills.

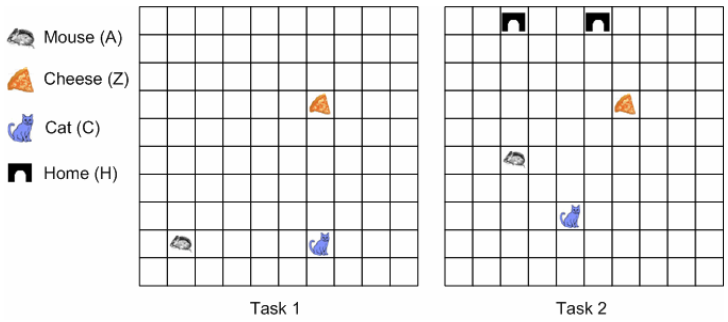

Fig. 2. Left: An example environment for task 1. Right: An example environment for task 2.

Experimental settings

Experiments of task 1 are carried out on randomly generated grid environments. The initial positions of the mouse, cat and cheese are randomly selected for each experiment. The mouse agent will receive a reward of +100 when it gets a cheese. If the cat catches the mouse, a punishment of -100 will be received by the agent, and the episode will end. Any other action taken by the mouse will get a step penalty of -1.

We first evaluate the performances of three different types of learning agents. The first type is the agents without skills. Agents without skills will perform Q-learning on the task and basically they seek to learn task 1 in its entirety. The second type is agents with skills, but their skills are not trained before performing the task. The third type is agents with trained skills. For each type of agent, we generate 20 randomly initialized experiments. The experiments are stopped at episode 30000 and the data is recorded. Figure 3 (left) illustrates the learning curve of the experiments.

To compare agents with different experiences of skills, our second experiment of task 1 involved equipping the agent with different proficiencies of trained skills. We ran the agent with no trained skills and then with trained skills of 200, 400, 600 and 800 episodes. Figure 3 (right) compares the learning curves of the experiments.

Experimental results

In this part, we show and explain the experiment results of task 1.

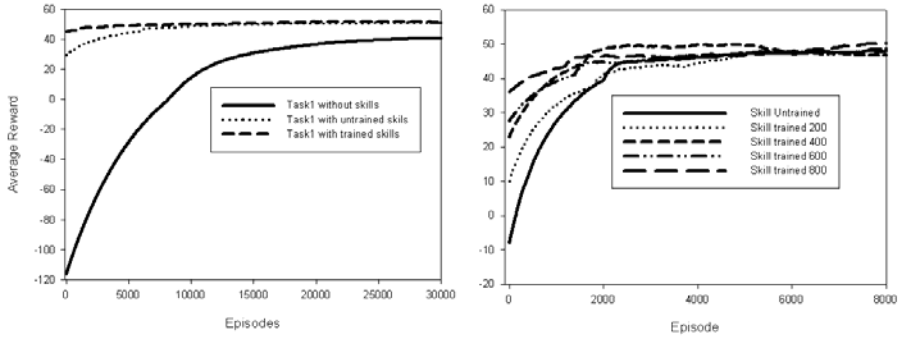

Fig. 3. Left: task 1 performance comparison of agents with combined skill, without skills and with trained skills. Right: task 1 performance comparison of agents with no trained skills and with trained skills of 200, 400, 600 and 800 episodes. All curves are smoothed using sampling proportion 0.2.

Figure 3 (left) compares the average learning curves of agents with combined skills, without skills and with trained skills. We can see from the left figure that agents with sufficient trained skills show significant better performance in task 1 than the other type of agents, especially at the early stages of learning (episodes 1- 5000). Agents using skills with no prior experience also show much better performance than agents without skills. Although these skills are not trained before applying to task 1, our skill combination method gives the agents a higher level of expression of the task. In a long run over 20000 episodes, agents with skills also gained more reward on average than standard learning agents. Figure 3 (right) compares average learning

curves of agents with different experiences of skills. This graph shows that gradually agents with more experience on skills perform better at the early state of learning. These experiments show that the skill combination method is stable for learning.

4.4 Task 2: Cheese Storage

Task 2 is more complex than task 1 in the cat and mouse domain (See figure 2 right). In this task, the mouse agent is required to get the cheese in the grid environment and then carry the cheese to the mouse's home. In the setting of task 2, an additional two homes - home1 (H_1) and home2 (H_2) are added into the environment. When a mouse is carrying a cheese, either of the two homes can be selected to store the cheese.

From this task description, we can define task 2 as follow:

$$GC(Z) \cap KA(C) \tag{4}$$

$$PC(H_1) \cap KA(C) \tag{5}$$

$$PC(H_2) \cap KA(C) \tag{6}$$

$$PC(H_1) \cup PC(H_2) \tag{7}$$

$$GC(Z) \rightarrow PC(H) \tag{8}$$

Expression (4) actually is the same as task 1. Expressions (5) and (6) represent concurrent skill combination of storing a cheese and keeping away from cat. Expression (7) represents the optional combination of storing cheese into home 1 or home 2. Expression (8) represents the sequential combination of getting the cheese and storing the cheese into a home. Hence, we can use knowledge reasoning methods to get the following expression of task 2:

$$(GC(Z) \rightarrow (PC(H_1) \cup PC(H_2))) \cap KA(C)$$

Experimental settings

We carried out experiments of task 2 on randomly generated grid worlds. The initial positions of mouse, cat, cheese and homes are randomly selected during each of the experiments. The mouse agent receives a medium reward of +10 when it gets a cheese, and when it successfully puts the cheese into one of the homes, it gets a positive reward of +100. If the cat catches the mouse, a punishment of -100 will be received by the agent, and the episode ends. Any other actions taken by the mouse gets a step penalty of -1.

As task 1, we did two types of experiments in task 2. The first type of experiment compares agents with skills, without skill and with trained skills. Figure 4 (left) shows the learning curve of the first type of experiments. The second type of experiment compares agents with skills that have different levels of proficiency. Figure 4 (right) shows the graph of results.

Experimental results

In this part, we present and explain the experimental results of task 2.

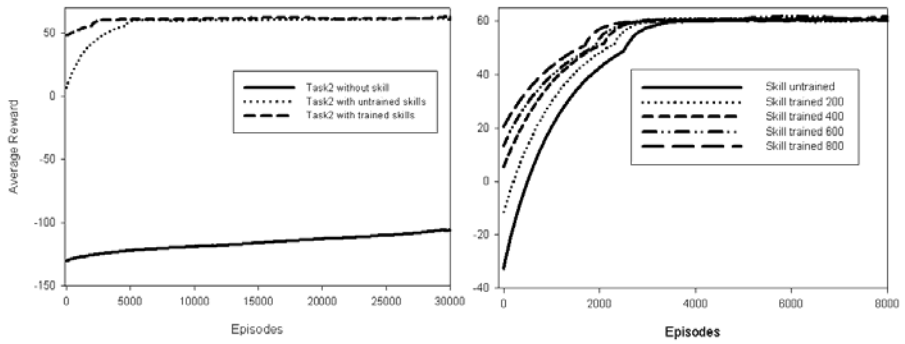

Fig. 4. Left: Task 2 performance comparison of agents with skills, without skill and with trained skills. Right: Task 2 performance comparison of agents with no trained skills and with trained skills of 200, 400, 600 and 800 episodes. All curves are smoothed using sampling proportion 0.2.

Figure 4 (left) compares average learning curves of agents with combined skills, without skills and with trained skills. In task 2, we find that agents using skills, no matter if they are trained or not, show remarkably better performance than agents without skills throughout the whole test (from episode 1-30000). Agents using skills with no prior experience reach a high standard of average reward (roughly +58) before episode 5000, while agents without skills still have a large amount of negative reward (roughly -122) at that stage. Even in a long run to episode 30000, agents without skills only make a small improvement to reward -106 which is far from optimal. Figure 4 (right) compares average learning curves of agents in task 2 with different skill experience. This graph shows that gradually agents with more experience on skills perform better at the early stage of learning. These experiments show that the skill combination method exhibits stable improvement for learning.

4.5 Compare Task 1 and Task 2

We will compare the learning efficiency of task 1 and task 2 in this section.

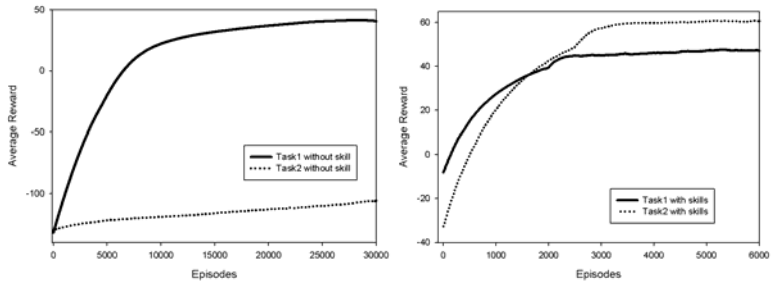

Fig. 5. Left: Compare learning curve of agents without skill between task1 and task2. Right: Compare learning curve of agents with skill between task1 and task2.

Figure 5 (left) compares the learning curves of agents without skills between task 1 and task 2. The graph shows that task 2 is a much more difficult problem compared to task 1. Although only 2 more features (Home 1 and Home 2) were added into task 2, the performance of the agent decreases dramatically. Agents without skills only improve nearly +1 more average reward in every 5000 episode of learning. On the other hand, task 2 is not a hard problem to agents with skills. Figure 5 (right) show that skilled agent performs very well in task 2. They reach a high standard of average reward between episodes 4000 to 5000. The learning curve of task 2 shows a steeper rise than task 1 and also gives a higher average reward when the learning is stable. The reason is that the settings of task 2 sets a medium reward of +10 when the agent gets cheese and +100 for the agent putting cheese to the home, so the sum of task 2's reward is +10 more than task 1's.

5 Discussion

From the previous section's comparison, we find that agents with skills show much better performance than agents without skills, especially when the problem becomes complex as in task 2. We use the big *O* notation to describe the asymptotic bound of the agent's state space and use *n* to denote the number of learning features in the task and *k* as the size of a learning feature. Then the scalability of state space of agents with skills can be denoted as $O(kn)$ which is a linear function of *n* and *k*. Agents use appropriate skills to deal with each of the learning features in the problem. The relations of these learning features are abstracted to a higher level which can be solved by combined skills. So the size rise of state space from task 1 to task 2 is a *linear growth*. On the other hand, the state space of agents without skill is $O(k^n)$. Every new learning feature added to the problem will make the state space *grow exponentially* in respect of the feature's size *k*. We know that $O(kn) \ll O(k^n)$ when $k>10$, $n>2$. So the learning efficiency of agent with skills is much better than agents without skills.

The trained skills in our work are reusable in the same problem domain. Trained skills are given to agents in task 1 and task 2 in the cat and mouse domain, and they generate better performance than an agent with no skill. Skills represent transferable knowledge to different tasks in the problem domain. In recent work, [7] proposes a method to construct mapping to value function based transfer in RL. Using policy based transfer, [8] utilizes transfer through inter-task mappings to construct a transfer functional from a source task to a target task. Another approach is to use shaping. [9] introduces the use of learned shaping reward in RL tasks, where an agent uses prior experience on a sequence of task to learn a portable predictor. These researches show that using transfer learning can markedly reduce learning time on different task.

Our work can also be viewed as a kind of model based learning method which uses tasks to represent problems at a high level and uses skills to construct a model of tasks. [6] provides an introduction of models and planning. A recent approach described in [10] uses a designed environment model to generate a selection of policy reinforcement learning.

Finally, we define principles for combining skills in RL, but we did not specify how these principles are implemented in different problem domain. We need a flexible representation at a high level, so that implementation can be carried out on further problems.

6 Conclusion and future work

In this paper, we present a skills combination method for high level knowledge representation in reinforcement learning. The experimental results and analysis shows that our method gives the learning agents a flexible way to express and solve different tasks. Our method can also reduce the learning space of the problems which significantly improves the learning speed of agent. In this study, agents use skills as predefined knowledge. But we believe that agent can automatically generate task representation. This is a topic for our future work.

References

1. Konidaris, G.D. and A.G. Barto, *Building Portable Options: Skill Transfer in Reinforcement Learning.* Proceedings of the Twentieth International Joint Conference on Artificial Intelligence 2007, Hyderabad, India, January 6-12, 2007, 2007.
2. Taylor, M.E. and P. Stone. *Cross-Domain Transfer for Reinforcement Learning.* in *In Proceedings of the Twenty-Fourth International Conference on Machine Learning, ICML07'*. 2007.
3. Puterman, M.L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 2005: Wiley-Interscience.
4. Russell, S. and P. Norvig, *Artificial Intelligence: A Modern Approach*. 2003: Prentice Hall Series in Artificial Intelligence. P763-788.
5. Watkins, C. and P. Dayan, *Q-Learning.* Machine Learning, 8(3-4):279--292, 1992, 1992.
6. Sutton, R. and A. Barto, *Reinforcement Learning: An Introduction*. 1998: MIT Press.
7. Liu, Y. and P. Stone. *Value-Function-Based Transfer for Reinforcement Learning Using Structure Mapping*. in *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. 2006.
8. Taylor, M.E., S. Whiteson, and a.P. Stone. *Transfer via InterTask Mappings in Policy Search Reinforcement Learning.* in *In The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, May 2007*. 2007.
9. Konidaris, G. and A. Barto. *Autonomous Shaping: Knowledge Transfer in Reinforcement Learning*. in *Proceedings of the Twenty Third International Conference on Machine Learning* 2006. Pittsburgh.
10. Kalyanakrishnan, S., P. Stone, and Y. Liu. *Model-based Reinforcement Learning in a Complex Domain*. in *RoboCup-2007: Robot Soccer World Cup XI, Springer Verlag, Berlin, 2008*. 2007.