The Thirteenth International Symposium on Artificial Life and Robotics 2008(AROB 13th '08),
B-Con Plaza, Beppu, Oita, Japan, January 31-February 2, 2008

Networked Reinforcement Learning

Makito Oku[*] Kazuyuki Aihara[†,‡,*]

[*]Department of Mathematical Informatics, Graduate School of Information Science and Technology,
The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

[†]Institute of Industrial Science, The University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan

[‡]Aihara Complexity Modelling Project, ERATO, Japan Science and Technology Agency (JST),
3-23-5-201 Uehara, Shibuya-ku, Tokyo 151-0064, Japan

e-mail: oku@sat.t.u-tokyo.ac.jp

Abstract

Recently, many models of reinforcement learning with hierarchical or modular structures have been proposed. They decompose a task into simpler sub-tasks and solve them with multiple agents. In these models, however, topological relations of agents are severely restricted. By relaxing the restrictions, we propose networked reinforcement learning where each agent in a network acts in parallel as if the other agents are parts of the environment. Although convergence to an optimal policy is no longer assured, we show by numerical simulations that our model performs well at least in some simple situations.

Key words hierarchical reinforcement learning, modular reinforcement learning, partially observable Markov decision process

1 Introduction

Reinforcement learning (RL) is a learning method in which an agent interacting with the environment updates its policy sequentially to maximise the cumulative reward by try and error [1]. Although RL is applicable to a variety of tasks, it has a problem called the curse of dimensionality: it does not work well for tasks with a large state space. To cope with this problem, hierarchical RL and modular RL have been proposed [2–8]. These models decompose a task into simpler sub-tasks and solve them with multiple agents.

However, these models have severe restrictions on topological relations of agents. First, these models cannot deal with cyclic dependencies among agents, because in such a structure task decomposition is difficult to consider analytically. Recurrent connections, however, are important for partially observable Markov decision processes (POMDPs) [9]. Next, most models require manual task decomposition, because it is difficult to assure convergence of learning if the role of each agent is not defined in advance. Automatic task decomposition has been succeeded only in special cases [5,7]. Furthermore, many models are not suitable for parallel computations. For example, models such as options [8], MAXQ [4], and HAM [6] work similarly to the sequential computer program: once a super-agent calls a sub-agent, the control moves to the sub-agent until the terminal conditions. Other models such as MMRL [5] use an integration mechanism which integrates information from all the agents.

On the other hand, in our brains an enormous number of neurons form a complex network and do information processing. The neural system is free from the restrictions described above: the network contains many cycles, the role of each neuron changes by learning, and neurons work in parallel. What is interesting is the neural system somehow works well to solve daily complex cognitive tasks.

What would occur if we relax the restrictions of a RL model with multiple agents? Can it work properly and flexibly like our brains? To answer the questions, we propose networked RL where agents in a network act in parallel as if the other nodes are parts of the environment. Our model cannot learn an optimal policy: an agent in a network knows neither all the ancestors' states [10] nor neighbour's policies [11]. However, some RL models can be reconstructed using networked RL with slight modifications. As examples, we reconstruct three representative models of hierarchical RL, modular RL, and POMDP [3,7,9]. Then, we solve the same tasks used in the original papers.

In this paper, we use 'reward' and 'payment' for incoming and outgoing reinforcement signals, respectively. We also use 'world' and 'environment' for outside of the whole system and outside of an agent, respectively.

2 Networked Reinforcement Learning

Networked RL consists of multiple agents, and their dependencies are shown by directed arrows. Thus, the whole system forms a network (Fig. 1a). When we focus on a single agent in networked RL, its situation is similar to that of standard RL. In standard RL, problems are modelled as Markov decision processes (MDPs, Fig. 1b). In MDPs, an agent receives a state from the world and takes an action to the world in each time step. Then it receives a real-valued reward from the world.

Similarly, an agent in networked RL receives a state from the world or its parent nodes, and it take an action to the environment (Fig. 1c). It also receives a reward from the environment. A crucial difference from MDPs is that the environment of each agent does not necessarily have the Markov property, even if the world itself has. Another difference is that an agent sends a reinforcement signal called payment to its child nodes. Notice that when there is only one node, the situation is equivalent to a MDP where a payment does not make sense.

The payment function depends on state s, action a, reward r, and next state s'. However, it is efficient to exclude r from the function. For example, if a child node knows only action a and payment p from its parent, and if payment p depends on r (thus, p also depends on s and s'), it is difficult for the child node to maximise p without knowing s and s'.

For simplicity, we do not update the payment function by learning. Instead, we determine it in advance. Let $\mathcal{A}$ and $\mathcal{S}$ denote sets of actions and states of an agent, respectively. For each action $a \in \mathcal{A}$, a set of target states $\mathcal{D}(a) \subset \mathcal{S}$ is defined in advance. If current state s is not a member of the target states $\mathcal{D}(a)$, a payment is sent only when a new state $s' \neq s$ is observed. The payment is positive if $s' \in \mathcal{D}(a)$ and negative if $s' \notin \mathcal{D}(a)$. On the other hand, if current state s is a member of the target states $\mathcal{D}(a)$, a positive payment is sent whenever s' stays in the domain. However, to avoid deadlocks among agents, a negative payment is sent when a time-out occurs.

We use standard Q-learning and the Boltzmann selection for each agent. A new action is selected only after a different state is observed or a time-out occurs. We also define two special actions called inhibit signals and null signals. If an agent gets more than one inhibit signal from its parent nodes, it stops updating its policy and sends null signals to all child nodes. These special signals are introduced for compatibility with modular RL models.

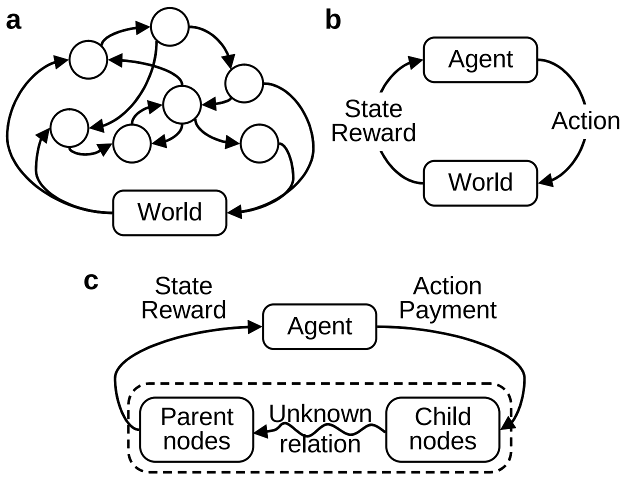

Fig. 1: Schematic diagrams of (**a**) networked RL, (**b**) Markov decision processes, and (**c**) interaction between an agent in a network and its environment. Payment means a rewarding signal sent to a child node.

3 Simulations

We reconstructed three representative RL models using networked RL. Figure 2 shows the network structures of the three reconstructed models.

First, feudal Q-learning[3] is a representative model for hierarchical RL. It consists of multiple layers of RL agents to deal with state representations of multiple resolution. It can speed up learning because global and local searches in the parameter space coexist. One fault is that it converges to a sub-optimal policy.

In the original model, an agent is assigned to every grid cell in each layer. However, this assumption is not natural. Thus, we modified to use an agent per layer, regarding a location of the grid world as a state.

Next, compositional Q-learning[7] is a representative model for modular RL. It aims for compositional tasks which are composed of several elemental tasks. Module agents learn elemental tasks, while the gating agent selects a task among the elemental tasks according to the 'augmenting bits' which deliver higher-order information.

A different point is that in the reconstructed model unselected agents do not learn at all, while in the original model modules learn in proportion to the prediction errors of the Q-values.

Finally, Cassandra *et al.*'s model[9] is a representative model of POMDPs. This model has a recurrent connection to deal with partial observability, because the internal state reflects whole the history of state observations.

In the original model, an agent called state estimator represents the probability distribution of the true state. And it has a connection to itself. In the reconstructed model, each agent uses discrete states and actions, and the first agent was decomposed into two.

Using the three reconstructed models, we solved the same tasks used in the original papers (Fig. 3). In the middle task, an agent has to visit at most three subgoals in the correct order. In the right task, an agent can only observe whether or not the current state is the goal.

The performance of the reconstructed models were compared with standard RL with a single agent. Parameters for learning were set to the same between standard RL and networked RL in each task except for parameters for the payment functions.

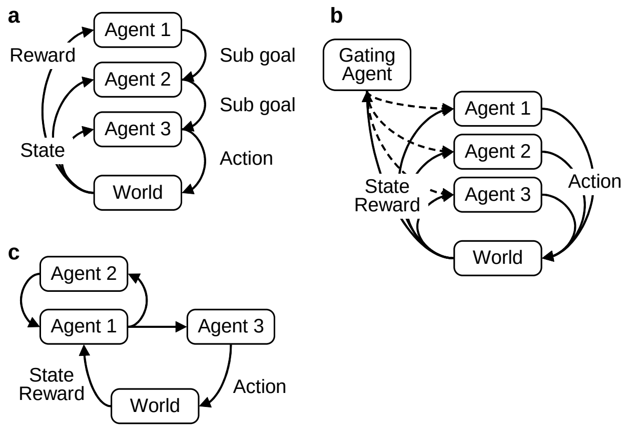

Fig. 2: Reconstructed network structures of (**a**) hierarchical model, (**b**) modular model, and (**c**) POMDP model.

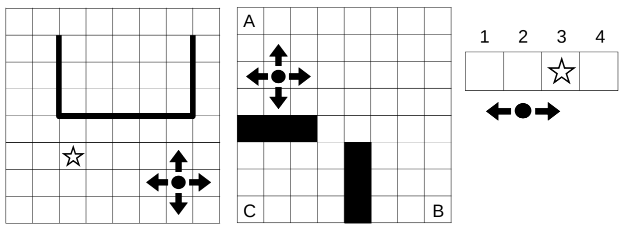

Fig. 3: Tasks used for (Left) hierarchical model, (Middle) modular model, and (Right) POMDP model, respectively. Black circles denote the current position in each grid world, and arrows show possible movements. Positions marked by a star are goals, and A, B, or C are sub-goals.

4 Results

Figure 4 shows the learning curves of the three simulations. All the models exhibited the same characteristics as the original models: early convergence and sub-optimality (top) and successful learning of optimal policies (middle and bottom).

After learning, agents worked cooperatively. They obeyed orders from the above (hierarchical model), separated sub-tasks (modular model), and stored previous observations (POMDP model), respectively.

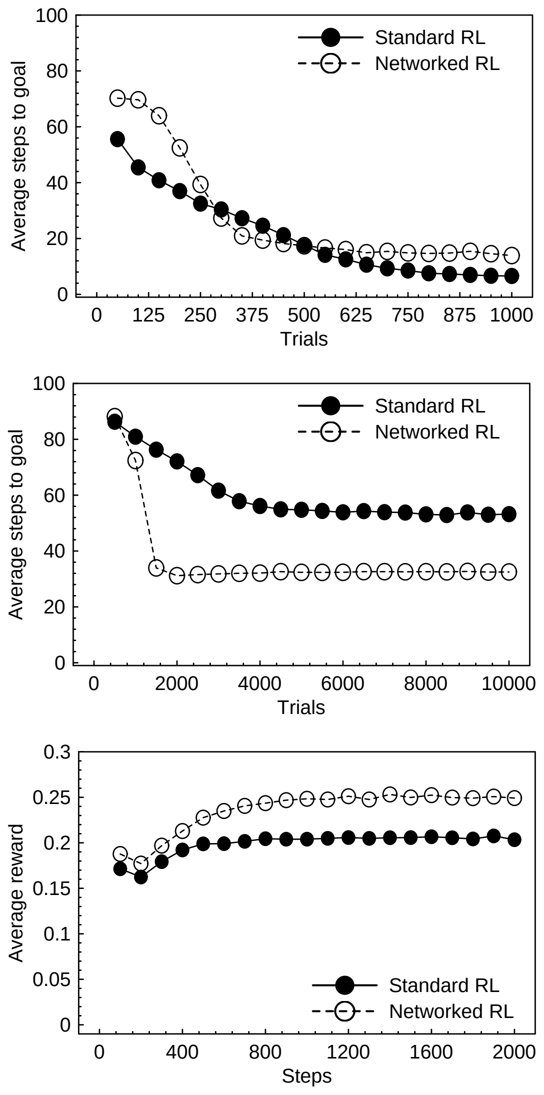

Fig. 4: Learning curves of simulations of (Top) hierarchical model, (Middle) modular model, and (Bottom) POMDP model. Each point shows an average value of 50, 50, and 100 trials, respectively.

5 Discussion

By relaxing restrictions, networked RL has both advantages and drawbacks compared with other hierarchical or modular RL models. In this paper, a few of them were studied.

First advantage of networked RL is that it can deal with cyclic dependencies among agents. This property is efficient for POMDPs, as the result of POMDP model showed. Furthermore, networked RL with many nodes and recurrent connections would have complex internal dynamics which is partially independent of the external world. Effects of internal dynamics to the learning performance will be interesting.

Next, networked RL is suitable for parallel computations, because each agent uses only local information. However, we updated agents synchronously in the test simulations.

Another property we did not study is flexibility such as automatic task decomposition. Since network structures used in the test simulations were reconstructed from already existing models, and since we set the payment functions manually, it is still an open question whether or not networked RL can work as flexible as our brains.

On the other hand, the main disadvantage of networked RL is that there is no assurance to converge to an optimal policy. However, all of the three reconstructed models performed well and showed similar characteristics to the original models. What we have to do next is to apply our model to much more complex tasks.

6 Summary

We have relaxed restrictions on hierarchical or modular reinforcement learning models to make them more similar to our brains, and proposed the networked RL where each agent in a network acts in parallel as if the other agents are parts of the environment. Although convergence to an optimal policy is no longer assured, we have shown that our model performed well at least in some simple situations by reconstructing three representative RL models using networked RL.

Acknowledgements

This study is partially supported by Grant-in-Aid No. 17022012 for Scientific Research on Priority Areas System study on higher-order brain functions from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Sutton RS, Barto AG (1998), *Reinforcement learning: An introduction*, MIT Press

[2] Bakker B, Schmidhuber J (2004), Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization, In *Proceedings of the 8-th Conference on Intelligent Autonomous Systems*, pp. 438–445

[3] Dayan P, Hinton GE (1993), Feudal reinforcement learning, In *Advances in Neural Information Processing Systems*, pp. 271–278

[4] Dietterich TG (2000), Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research* 13:227–303

[5] Doya K, Samejima K, Katagiri K, *et al.* (2002), Multiple model-based reinforcement learning, *Neural Computation* 14(6):1347–1369

[6] Parr R, Russell S (1998), Reinforcement learning with hierarchies of machines, In *Advances in Neural Information Processing Systems*

[7] Singh SP (1992), Transfer of learning by composing solutions of elemental sequential tasks, *Machine Learning* 8(3–4):323–339

[8] Sutton RS, Precup D, Singh S (1999), Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, *Artificial Intelligence* 112(1–2):181–211

[9] Cassandra AR, Kaelbling LP, Littman ML (1994), Acting optimally in partially observable stochastic domains, In *Proceedings of the Twelfth National Conference on Artificial Intelligence*

[10] Dolgov D, Durfee E (2004), Graphical models in local, asymmetric multi-agent Markov decision processes, In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 956–963

[11] Nair R, Varakantham P, Tambe M, *et al.* (2005), Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs, In *Proceedings of the Twentieth National Conference on Artificial Intelligence*