Utilizing Dynamics and Reward Models in Learning Strategy Fusion

Akihiko Yamaguchi* **, Jun Takamatsu*, and Tsukasa Ogasawara*

* Nara Institue of Science and Technology
** JSPS Research Fellow
{akihiko-y, j-taka, ogasawar}@is.naist.jp

Learning strategy (LS) fusion is our previous work in reinforcement learning (RL) framework. LS fusion fuses multiple LSs, such as transfer learning, for a single task of a robot. This paper introduces two LSs into LS fusion: an LS to learn a dynamics and a reward models, and an LS using a model-based RL method. Especially, we propose to use the MixFS dynamics model which is also our previous work. MixFS decomposes the dynamics model into the task specific elements and the task invariant elements. Thus, we can initialize the dynamics model of a task by transferring the one of the other task. In simulation experiments, we apply LS fusion with the new LSs to maze tasks of a small humanoid robot, where the primitive motions, crawling and turning, are also pre-learned by LS fusion. The results demonstrate that the new LSs improve the learning speed by using MixFS.

Key Words: Reinforcement Learning, Learning Strategy Fusion, Dynamics Model, Planning

1. Introduction

Designing a behavior by only its objective is essential for future robots, since this ability enables the end-users to teach their wish to the robots easily. We study reinforcement learning (RL) method as such a technology. There are a number of RL applications to robotics research [1, 2, 3]. However, RL methods require a lot of learning cost in large domains, such as motion learning of a humanoid robot. To overcome this issue, many *learning strategies (LSs)* are proposed to improve RL; for instance, dimension reduction [1].

Yamaguchi *et al.* proposed a method to fuse such LSs, named LS fusion [4]. Its key feature is the automatic LS-sequence selection; *multiple* LSs are applied to each task of a single robot *multiple* times, where the ordering of the LSs is automatically decided. For LS fusion, a modularized learning system was developed as shown in Fig. 1. Though we can use any LS with LS fusion, especially, transfer learning methods [5] work well with LS fusion as demonstrated in [4].

Therefore, this paper introduces the following LSs:

LS-model generates a dynamics and a reward model modules. Especially, we utilize MixFS [6] as the dynamics model to transfer the model parameters between tasks.

LS-planning generates a behavior module that uses the dynamics and the reward model modules and executes planning to acquire a policy.

Here, the dynamics model MixFS proposed by Yamaguchi *et al.* can decompose the dynamics model of the robot into the task specific elements and the task invariant ones [6]. Thus, the dynamics model module for a new task can be generated by transferring the dynamics model module that is previously learned for the other task. It is considered that this transfer improves the learning speed of the new task.

In the simulation experiments, we test the transfer of the dynamics model between two maze tasks in a simple and a complex setups. The simple task setup uses an omniwheel mobile robot in 2-dimensional plane, and the complex one uses a small humanoid where the primitive motions, crawling and turning, are pre-learned by using LS fusion. The results show that the LS-model and the LS-planning improves the learning speed by using MixFS.

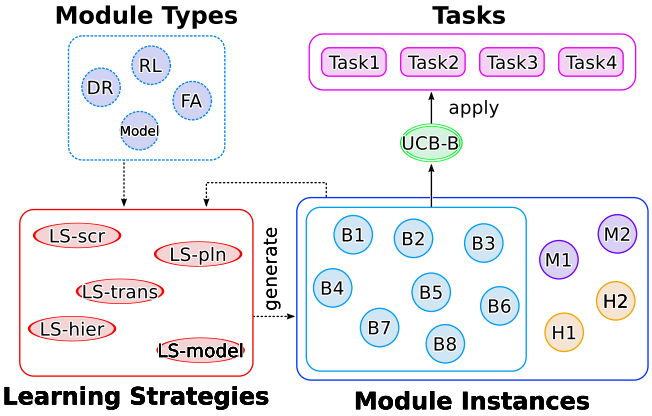

Fig.1 Overview of the modularized learning system. Every small circles, ellipses, and rectangles indicate modules. UCB-B denotes UCB-Boltzmann selection method.

The rest of this paper organized as follows. Section 2. describes LS fusion. Section 3. proposes the LS-model and the LS-planning. Section 4. describes the experiments. Finally, Section 5. concludes this paper.

2. Learning Strategy Fusion

LS fusion has two key elements: (1) the LSs, and (2) UCB-Boltzmann selection. The LSs generate modules, and UCB-Boltzmann selection chooses a *behavior module* that actually decides the behavior of the robot and learns the policy. This paper assumes that there are two types in LSs; *behavior* LSs that generate new behavior modules, and *supplementary* LSs that generate supplementary modules other than behavior modules, such as model modules and hierarchical-action-space modules. Let $\mathcal{LS}_{\mathrm{bhv}}$ denote

the set of the behavior LSs, $\mathcal{LS}_{\rm spl}$ denote the set of the supplementary LSs. Each behavior learning strategy LS is defined as a function $\mathsf{GEN}_{\rm bhv}(LS, \mathcal{U}, \mathcal{X}, Task)$ that generates behavior modules, and each supplementary learning strategy is defined as a function $\mathsf{GEN}_{\rm spl}(LS, Task)$ that generates supplementary modules. UCB-Boltzmann selection method chooses a behavior module from both the existing behavior modules and the new ones generated by the behavior LSs.

In LS fusion, the following procedure is repeated.

(1) The supplementary LSs generate modules if applicable.
(2) The behavior LSs generate new behavior modules.
(3) UCB-Boltzmann selection method chooses a behavior module.
(4) Several episodes are performed using the selected behavior module, and the policy of the behavior module is updated from samples. The supplementary modules are also updated if possible.

Here, the UCB (upper confidence bound) uses both the mean of a reward summation $\overline{R}_B$ and its deviation $\overline{\sigma}_B$. The deviation $\overline{\sigma}_B$ can estimate the potential improvement of the performance. The deviation $\overline{\sigma}_B$ is also used to judge if a behavior module is trained enough.

The complete algorithm is as shown in Algorithm 1. This algorithm is defined for an episodic task. We assume that several pairs of a control command space and a state space $\{(\tilde{\mathcal{U}}, \mathcal{X})\}$ are predefined; they have different DoF (Degree of Freedom) configurations. $\mathrm{N}_{\rm LSSp}$ is an interval of executing the supplementary LSs (LSSp means LS Supplementary), $\mathrm{N}_{\rm LSBh}$ is an interval of executing the behavior LSs (LSBh means LS Behavior). $\mathrm{N}_{\rm LSBh} > 1$ is needed to compute the valid reward statistics (we choose $\mathrm{N}_{\rm LSSp} = 20$ and $\mathrm{N}_{\rm LSBh} = 10$ in the experiments of this paper). UCB-Boltzmann selection method chooses a behavior module from both the existing $\mathcal{B}$ and new behavior modules generated by the behavior LSs. Note that only the selected new behavior module is added into $\mathcal{B}$.

Thus, the key elements of LS fusion are each LS ($\mathsf{GEN}_{\rm bhv}$, $\mathsf{GEN}_{\rm spl}$) and UCB-Boltzmann selection method. Note that LS fusion works with any LS for that $\mathsf{GEN}_{\rm bhv}$ or $\mathsf{GEN}_{\rm spl}$ is defined. The rest of this section describes UCB-Boltzmann selection method. In the next section, we specify the LSs used in this paper.

2.1 UCB-Boltzmann Selection

We evaluate the performance of a behavior module by $R \triangleq \frac{\sum_t r_t}{T}$, where $\{r_t | t = 1, 2, \dots\}$ denotes the observed reward sequence in an episode, and T denotes total time in the episode. Since each behavior module is updated to obtain better policy, its performance changes with episodes. Thus, we compute the mean and the standard deviation of R with forgetting the old data. Let $R_{N_{\rm eps}}$ the observation at an $N_{\rm eps}$-th episode. The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are updated by

$$\overline{R}_B \leftarrow \alpha_{\rm R} R_{N_{\rm eps}} + (1 - \alpha_{\rm R})\overline{R}_B, \quad (1)$$

$$\overline{R^2}_B \leftarrow \alpha_{\rm R} R^2_{N_{\rm eps}} + (1 - \alpha_{\rm R})\overline{R^2}_B, \quad (2)$$

where $\alpha_{\rm R}$ is a learning rate. The standard deviation of R can be obtained by $\overline{\sigma}_B = (\overline{R^2}_B - \overline{R}^2_B)^{1/2}$. In addition,

Algorithm 1: Learning strategy fusion

```
Input: Task Task, behavior modules B,
       state-space modules {X},
       action-space modules: {U},
       dynamics-model modules {M_dyn},
       reward-model modules {M_rwd}
       /* {M_dyn} and {M_rwd} may be empty */
for N_eps = 1, 2, ... do /* N_eps: episode number */
  if N_eps mod N_LSSp = 0 then
    for each LS ∈ LS_spl do
      {X}, {U}, {M_dyn}, {M_rwd}
          ← GEN_spl(LS, Task)
  if N_eps mod N_LSBh = 0 then
    /* select a behavior module: */
    B_new ← {}
    for each (U, X) do
      for each LS ∈ LS_bhv do
        B_new ← B_new ∪ GEN_bhv(LS, U, X, Task)
    Select B_next from B ∪ B_new by UCB-Boltzmann
    selection
    if B_next ∈ B_new then B' ← B ∪ {B_next} else
    B' ← B
    return B_next, B'
  Perform the episode with B_next:
    B_next is updated by its own learning algorithm
    {M_dyn}, {M_rwd} are updated if possible
  Update the reward statistics R̄_{B_next}, R̄²_{B_next},
  σ̄_{max B_next}
```

at the end of each episode, $\overline{\sigma}_{\max B}$ is updated for some LSs. The reward statistics $\overline{R}_B$, $\overline{R^2}_B$ are initialized by zero if B is generated by the LS-planning.

The UCB of R is defined by

$$R_{{\rm UCB}B} \triangleq \overline{R}_B + \mathrm{f}_{\rm UCB}\overline{\sigma}_B \quad (3)$$

where $\mathrm{f}_{\rm UCB}$ is a real constant value that decides the weight of expected improvement (typically 1 or 2).

According to Boltzmann selection, the probability to select B is defined as

$$\pi(B) \propto \exp\left(\frac{1}{\tau_{\rm lsd}} R_{{\rm UCB}B}\right) \quad (4)$$

where $\tau_{\rm lsd}$ is a temperature parameter to adjust randomness. We decrease $\tau_{\rm lsd}$ with $\tau_{\rm lsd} = \tau_{\rm lsd0} \exp(-\delta_{\tau_{\rm lsd}} N_{\rm eps})$.

3. Learning Strategies

This section defines the LSs, namely, defines a function $\mathsf{GEN}_{\rm bhv}(LS, \mathcal{U}, \mathcal{X}, Task)$ or $\mathsf{GEN}_{\rm spl}(LS, Task)$ for each learning strategy LS. The LS-Scratch, the LS-Accelerating, the LS-Freeing, and the LS-Hierarchy are as described in [4].

3.1 LS-Planning

The function $\mathsf{GEN}_{\rm bhv}(\mathsf{LS\text{-}pln}, \mathcal{U}, \mathcal{X}, Task)$ generates a behavior module if $\mathcal{U}$ is a discrete action space, and a dynamics and a reward model modules for $\mathcal{U}, \mathcal{X}, Task$ are available. The generation with the same setup is prevented since the probability that the new behavior module obtains better performance than the existing one is not high.

A new behavior module uses Dyna-MG algorithm proposed by Sutton *et al.* [7]. This algorithm combines a model-based and a model-free RL algorithms.

As the model-based RL algorithm, Dyna-MG utilizes the Improved Prioritized Sweeping algorithm proposed by McMahan and Gordon [8]. Though the original Dyna-MG [7] uses Q(0)-learning, we use Q(λ)-learning because of its learning-speed advantage. The both algorithms learn a same linear-function-approximator for the action value function. For this approximator, a predefined set of BFs (Basis Functions) is employed.

3.2 LS-Model

The function $\mathsf{GEN}_{\mathrm{spl}}(\mathsf{LS\text{-}model}, \mathit{Task})$ generates a dynamics model and a reward model for each combination of $(\mathcal{U}, \mathcal{X})$ and Task where $\mathcal{U}$ is a discrete action space. The original Dyna-MG algorithm [7] uses a simple linear model on the feature space for each model; here, the feature space is produced from the state space by the BFs. Thus, this dynamics model is assumed to be a linear model of the feature space dynamics. Rather than the feature space dynamics model, this paper uses an extended dynamics model proposed by Yamaguchi *et al.* [6]. This dynamics model is a composite model of the feature space dynamics and the state space dynamics, referred to as MixFS. Some model parameters of MixFS encode the task invariant elements. Thus, we can use such elements obtained in other tasks as prior knowledge.

The linear models on the feature space for the feature space dynamics and the reward are defined as follows:

$$\phi' \approx F_a \phi(x), \tag{5}$$

$$R \approx b_a^\top \phi(x), \tag{6}$$

where $\phi(x)$ denotes the feature vector (output of BFs $\mathcal{K}$) at state $x \in \mathcal{X}$, ϕ' denotes the succeeding feature vector by taking action a at x, and R denotes the reward obtained by this action. F_a and b_a are parameter matrices learned by an on-line gradient descent algorithm [9]. MixFS dynamics model is defined as follows:

$$\phi' \approx \delta F_a \phi(x) + \phi(x + \delta \tilde{x}_a(x)), \tag{7}$$

$$\delta \tilde{x}_a(x) = A_a x + B_a \phi(x) + d_a. \tag{8}$$

Here, $\delta F_a \phi(x)$ is a linear model of the feature space dynamics, and $x + \delta \tilde{x}_a(x)$ is a linear model of the state space dynamics. $\delta F_a \in \mathbb{R}^{|\mathcal{K}| \times |\mathcal{K}|}$, $A_a \in \mathbb{R}^{\dim(\mathcal{X}) \times \dim(\mathcal{X})}$, $B_a \in \mathbb{R}^{\dim(\mathcal{X}) \times |\mathcal{K}|}$, $d_a \in \mathbb{R}^{\dim(\mathcal{X}) \times 1}$ are the model parameters. Note that the parameters A_a and d_a encode the linear dynamics on the state space; thus, they may be task invariant. We use an on-line gradient descent algorithm to train the model parameters (see [6]).

After a model module is generated, the module is updated when an action in $\mathcal{U}$ is executed even if any behavior module does not use the model module. If there are some dynamics model modules learned in the other tasks, the new dynamics module is initialized using the parameters of the existing dynamics modules. Specifically, A_a and d_a are copied from the existing modules. If the state and the reward of a goal (or a forbidden region) are known, we can embed them on the reward model as prior knowledge. The specific method is described in [6].

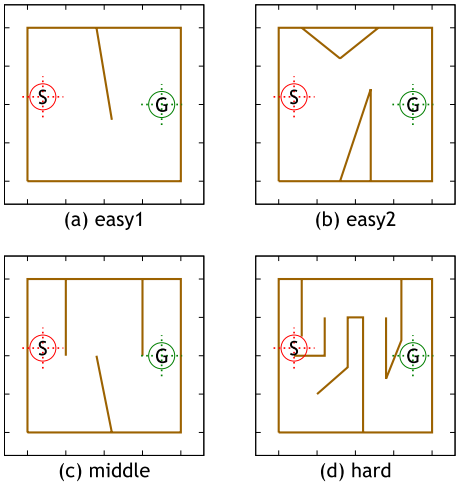

Fig.2 Types of mazes.

4. Experiments

This section demonstrates two simulation experiments of LS fusion. Especially, we verify the LS-planning and the LS-model. In the both experiments, two maze tasks are learned by a robot. Between these tasks, the LS-model reuses the task invariant parameters of the dynamics model. In the first experiment, we use an omniwheel mobile robot on a 2-dimensional plane [3]. In the second experiment, we use a small humanoid robot where the primitive motions, crawling and turning, are pre-learned by using LS fusion (see [4] for detail). There are four types of mazes, referred to as easy1, easy2, middle, and hard, as shown in Fig. 2.

4.1 Model Transfer in 2D-Maze Task

To demonstrate the LS-model and the LS-planning work as expected, LS fusion is applied to the maze task. The environment is the same as in [3]. The type of the maze changes at 400-th episode which means that two tasks are learned in sequence. Concretely, in 0 to 399-th episode, the maze middle is learned, then the maze hard is learned.

In this learning, each LS is expected to be used in the following scenario:

(1) The LS-model generates a dynamics and a reward model modules for the maze middle.
(2) The LS-planning generates a behavior module for the maze middle, where the model modules are used.
(3) The above modules are learned during 0 to 399-th episode.
(4) At the beginning of the 400-th episode, the LS-model generates a dynamics and a reward model modules for the maze hard. In this case, the dynamics model for the maze middle is used to initialize the parameters of the new model.
(5) The LS-planning generates a behavior module for the maze hard, where the model modules are used.
(6) The modules are learned.

Fig. 3 shows the resulting learning curves of this task (the mean of the return over 25 runs is plotted per episode). Here, the following methods are compared; LSF (MixFS): LS fusion where the LS-model uses MixFS dynamics model, LSF (Simple): LS fusion where the LS-model uses the Simple dynamics model,

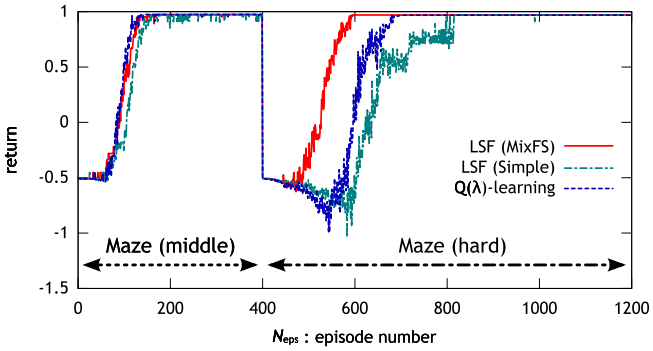

Fig.3 Resulting learning curves of the maze task where the maze changes at 400-th episode. Each curve shows the mean of the return over 15 runs per episode.

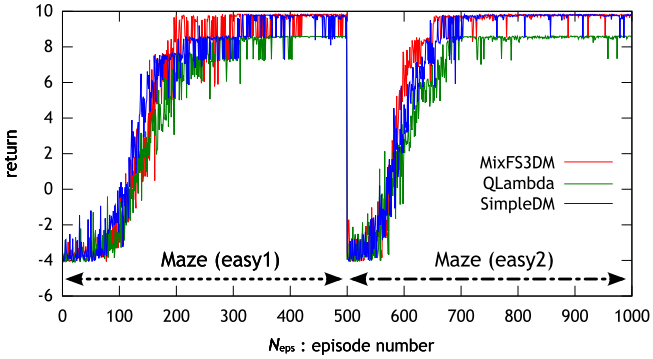

Fig.4 Resulting learning curves of the HumanoidMaze task where the maze changes at 1000-th episode. Each curve shows the mean of the return over 10 runs per episode.

and Q(λ)-learning: normal RL methods for the two tasks. In learning the maze middle, the learning speed of these three are almost the same. In learning the maze hard, LSF (MixFS) is faster than the others. A possible reason is that LSF (MixFS) uses the prior knowledge obtained in the maze middle by transferring the dynamics model. Anyway, LS fusion works as expected.

4.2 Model Transfer in Humanoid-Maze Task

This experiment demonstrates the availability of the LS-model and the LS-planning in the humanoid robot case. The task setup is similar to the one in the previous experiment. The differences are using a simulated humanoid robot and using pre-learned policies, crawling and turning, as the primitive actions. The detail of learning these primitive actions is as describes in [4]. In 0 to 499-th episode, a maze task (easy1) is learned, then a maze task (easy2) is learned. The same scenario as the previous experiment is expected in this learning.

Fig. 4 shows the resulting learning curves of this task (the mean of the return over 10 runs is plotted per episode). Here, the same three methods as the previous experiments are compared. In learning the maze easy1, the learning speed of these three are almost the same. In learning the maze easy2, LSF (MixFS) is slightly faster than the others due to transferring the dynamics model. The reason why the improvement of the learning speed is not large is considered to be using complex primitive actions. Fig. 5 shows the snapshots of an acquired behavior after the maze easy2 task.

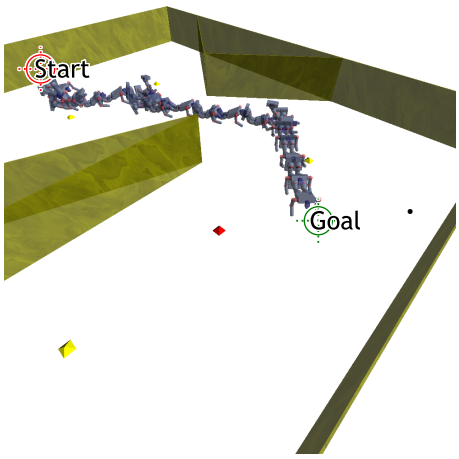

Fig.5 Snapshots of an acquired behavior after the maze easy2 task (taken in 1-FPS).

5. Conclusion

This paper introduced two learning strategies (LSs) into LS fusion [4]: the LS-model to generate a dynamics and a reward model modules, and the LS-planning to generate a behavior module that plans the policy for a task using the models. Especially, we utilized MixFS [6] as the dynamics model to transfer the model parameters between tasks. The simulation experiments were performed in maze tasks of both a simple omniwheel mobile robot and a small humanoid robot. The simulation results demonstrated that using MixFS improved the learning speed by transferring the model parameters.

Acknowledgements Part of this work was supported by a Grant-in-Aid for JSPS, Japan Society for the Promotion of Science, Fellows (22·9030).

Bibliography

[1] J. Morimoto, S. Hyon, C. Atkeson and G. Cheng: "Low-dimensional feature extraction for humanoid locomotion using kernel dimension reduction", the IEEE Internactional Conference in Robotics and Automation (ICRA'08), pp. 2711–2716 (2008).
[2] J. Kober and J. Peters: "Learning motor primitives for robotics", the IEEE Internactional Conference in Robotics and Automation (ICRA'09), pp. 2509–2515 (2009).
[3] A. Yamaguchi, J. Takamatsu and T. Ogasawara: "Constructing action set from basis functions for reinforcement learning of robot control", the IEEE Internactional Conference in Robotics and Automation (ICRA'09), Kobe, Japan, pp. 2525–2532 (2009).
[4] A. Yamaguchi, J. Takamatsu and T. Ogasawara: "Fusing learning strategies to learn various tasks with single configuration", IEICE Tech. Rep., Vol. 110 of NC2010-154, Tokyo, Japan, pp. 159–164 (2011).
[5] L. Torrey and J. Shavlik: "Transfer learning", Handbook of Research on Machine Learning Applications (Eds. by E. Soria, J. Martin, R. Magdalena, M. Martinez and A. Serrano), IGI Global, chapter 11 (2009).
[6] A. Yamaguchi, J. Takamatsu and T. Ogasawara: "Composition of feature space and state space dynamics models for model-based reinforcement learning", IEICE Tech. Rep., Vol. 109 of NC2009-8, Nara, Japan, pp. 7–12 (2009).
[7] R. S. Sutton, C. Szepesvári, A. Geramifard and M. Bowling: "Dyna-style planning with linear function approximation and prioritized sweeping", Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, pp. 528–536 (2008).
[8] H. B. McMahan and G. J. Gordon: "Generalizing dijkstra's algorithm and gaussian elimination for solving mdps", Technical Report CMU-CS-05-127, Carnegie Mellon University (2005).
[9] C. M. Bishop: "Pattern Recognition and Machine Learning", Springer (2006).