社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Composition of Feature Space and State Space Dynamics Models for Model-based Reinforcement Learning

Akihiko YAMAGUCHI†, Jun TAKAMATSU†, and Tsukasa OGASAWARA†

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, JAPAN
E-mail: †{akihiko-y,j-taka,ogasawar}@is.naist.jp

Abstract Learning a dynamics model and a reward model during reinforcement learning is a useful way, since the agent can also update its value function by using the models. In this paper, we propose a general dynamics model that is a composition of the feature space dynamics model and the state space dynamics model. This way enables to obtain a good generalization from a small number of samples because of the linearity of the state space dynamics, while it does not lose the accuracy. We demonstrate the simulation comparison of some dynamics models used together with a Dyna algorithm.

Key words Model-based reinforcement learning, Dyna-style planning, prioritized sweeping, dynamics model.

1. Introduction

A reinforcement learning (RL) architecture, named Dyna, where the model-free RL and the model-based RL are combined is proposed by Sutton [1]. The convergence of the Dyna with a linear function approximator is proven by Sutton *et al.* [2]. The setting of the problem in the Dyna is the same as RL, that is, to acquire an optimal decision making rule (policy) with an incompletely known dynamics of the environment in an on-line manner. In the Dyna architecture, a model-free RL, learning models, and the model-based RL (planning, in other words) are performed simultaneously. A key idea in the Dyna architecture is that if the model is learned, samples needed to train a value function can be withdrawn from the model. This makes the learning efficient. Although the Dyna architecture is a general method where a model-based and a model-free RL are combined, there are some similar approaches that utilize a model [3]∼[5].

One important problem in the Dyna architecture or the other model-based RLs is how to make a precise dynamics model. In this paper, we develop a composite dynamics model of the feature space dynamics and the state space dynamics. Here, the feature means the output of the basis functions (we are assuming to be local models) that are used to approximate a value function. Usually, the dynamics model is represented by a transition matrix of the feature. This dynamics model in the *feature space* is accurate, but has a poor generalization. Instead, the dynamics model in the *state space* may obtain a good generalization since in many practical applications, the dynamics of the environment (including the robot) is nearly linear in the state space. Typical examples are navigation tasks and robotic manipulations. Thus, our proposed composite dynamics model, which we call the MixFS dynamics model, can exploit the advantages of both dynamics models (Fig. 1).

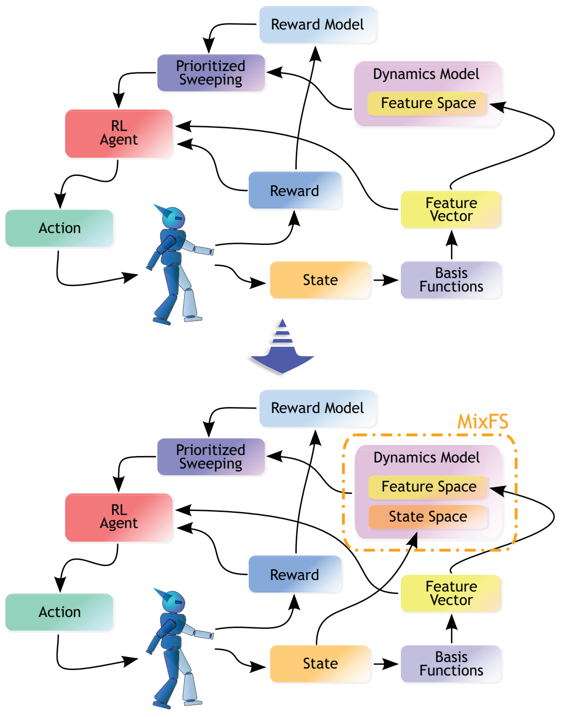

Fig. 1 The system architecture where the Dyna is utilized (above). In this paper, we develop a composite dynamics model of the feature space dynamics and the state space dynamics, named MixFS (below).

Our main contributions are (1) to combine the two dynamics models and make it work with the Dyna algorithms, and (2) to derive an on-line learning method of the developed model. In a simulation, we compare the MixFS dynamics model to a conventional linear dynamics model of a feature space in an environment that has a simple dynamics. The result of learning dynamics model demonstrates that the MixFS dynamics model can obtain an accurate model from a small number of samples, that is, it has a good generalization. Moreover, it can acquire a high accurate model from a large number of samples. We also compared these two models used with a Dyna algorithm in a navigation task on a 2-dimensional continuous maze. The result demonstrates that the Dyna with the MixFS is much faster than that with the conventional dynamics model.

2. Dyna Architecture with Linear Function Approximator

2.1 Reinforcement Learning with Linear Function Approximator

The purpose of RL is that a system (agent) whose input is a state, $x_n \in \mathcal{X}$, and a reward, $R_n \in \mathbb{R}$, and whose output is an action, $a_n \in \mathcal{A}$, acquires the policy, $\pi(x_n) : \mathcal{X} \to \mathcal{A}$, that maximizes the expected discounted return, $\mathbb{E}\big[\sum_{k=1}^{\infty} \gamma^{k-1} R_{n+k}\big]$, where $n \in \mathbb{N} = \{0, 1, \dots\}$ denotes the time step and $\gamma \in [0, 1)$ denotes a discount factor. In some RL methods, such as Q-learning [6] and Sarsa [7], an action value function, $Q(x, a) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, is learned to represent the expected discounted return by taking an action a from a state x. Then, the optimal action rule is obtained from the greedy policy $\pi(x) = \arg\max_a Q(x, a)$.

If $\mathcal{X}$ or $\mathcal{A}$ are continuous, a function approximator is used. Here, we assume to use a linear function approximator over a continuous $\mathcal{X}$ and a discrete $\mathcal{A}$. An action value function is represented as follows:

$$Q(x, a) = \theta_a^\top \phi(x) \tag{1}$$

$$\phi(x) = (\phi_1(x), \dots, \phi_{|\mathcal{K}|}(x))^\top \tag{2}$$

where θ_a and $\phi(x)$ denote a parameter vector and a feature vector respectively. $\mathcal{K} = \{\phi_k | k = 1, 2, ..\}$ denotes predetermined basis functions. In this paper, we use a Normalized Gaussian Network (NGnet) that is often used in RL applications [8], [9]. A basis function of the NGnet is defined as follows:

$$\phi_k(x) = \frac{G(x; \mu_k, \Sigma_k)}{\sum_{k'} G(x; \mu_{k'}, \Sigma_{k'})} \tag{3}$$

where $G(x; \mu, \Sigma)$ denotes a Gaussian with mean μ and covariance matrix Σ.

2.2 Linear Dyna with 'MG' Prioritized Sweeping

McMahan and Gordon proposed the Improved Prioritized Sweeping algorithm that is a fast planning algorithm in a Markov Decision Process (MDP) [10]. Sutton *et al.* called the algorithm *MG prioritized sweeping*, and developed a Dyna using it, *Dyna-MG*, which was faster than Dyna algorithms with the other prioritized sweeping methods [2]. Thus, we use the Dyna-MG whose algorithm is given as follows:

Algorithm 1: Dyna-MG

```
Input: a state set $\mathcal{X}$ (continuous), an action set $\mathcal{A}$ (discrete)
       basis functions $\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{K}|\times 1}$
Output: a coefficient vector $\theta_a$ for which $\tilde{Q}(x,a) \approx \theta_a^\top \phi(x)$
1: Initialize: $\theta_a, F_a, b_a$
2: for $N_{\text{eps}} = 1, 2, \dots$ do /* $N_{\text{eps}}$: episode */
3:   $n \leftarrow 1$ /* time index */
4:   Choose a start state $x_n \in \mathcal{X}$
5:   $\phi_n \leftarrow \phi(x_n)$
6:   while not is-end-of-episode($x_n$) do
7:     Carry out an action $a_n$ according to current policy,
         producing a reward $R_n$ and next state $x_{n+1}$
8:     $\phi_{n+1} \leftarrow \phi(x_{n+1})$
9:     Update by Q(0)-learning:
10:      $\delta_n \leftarrow R_n + \gamma \max_a \theta_a^\top \phi_{n+1} - \theta_{a_n}^\top \phi_n$ /* TD error */
11:      $\theta_{a_n} \leftarrow \theta_{a_n} + \alpha \phi_n \delta_n$
12:    Update models:
13:      $F_{a_n} \leftarrow F_{a_n} + \alpha(\phi_{n+1} - F_{a_n}\phi_n)\phi_n^\top$
14:      $b_{a_n} \leftarrow b_{a_n} + \alpha(R_n - b_{a_n}^\top \phi_n)\phi_n$
15:    for $i = 1, \dots, |\mathcal{K}|$ do
16:      if $\phi_{n[i]} > Th_1$ then
17:        Put $i$ on the PQueue with priority $|\delta_n \phi_{n[i]}|$
18:    for $1, \dots, N_{\text{pln}}$ do
19:      if PQueue is empty then break
20:      $i \leftarrow$ PQueue.pop()
21:      for all $(j, a) \in pred(i)$ do
22:        $\delta \leftarrow b_{a[j]} + \gamma \max_{a'} \theta_{a'}^\top (F_a \mathbf{e}_j) - \theta_{a[j]}$
23:        $\theta_{a[j]} \leftarrow \theta_{a[j]} + \alpha\delta$
24:        Put $j$ on the PQueue with priority $|\delta F_{a[i,j]}|$
25:    $n \leftarrow n + 1$
```

Here, $\{F_a, b_a | a \in \mathcal{A}\}$ denotes the parameters of the dynamics model and the reward model used as follows:

$$\phi' \approx F_a \phi(x) \tag{4}$$

$$R \approx b_a^\top \phi(x). \tag{5}$$

$N_{\text{pln}} \in \mathbb{N}$ denotes a planning depth (given constant), *PQueue* denotes a priority queue whose *pop*() operator removes and returns the queue element of the highest priority, $Th_1 \in [0, 1]$ is a threshold that decides to include the *PQueue*, $\mathbf{e}_j \in \mathbb{R}^{|\mathcal{K}|\times 1}$ denotes a vector whose j-th element is 1 and the rest are 0, $\square_{[i]}$ denotes the i-th element of the vector $\square$, $\square_{[i,j]}$ denotes the (i, j)-th element of the matrix $\square$. *pred*(i) denotes the set of all pairs of a feature index and an action (j, a) such that taking action a from the feature index j has a positive chance to reach the feature index i. In this paper, we define the $\{(j, a)\} = pred(i)$ as whole pairs that satisfy all of the following conditions:

$$j \in \{1, \ldots, |\mathcal{K}|\}, \qquad i \neq j, \tag{6}$$

$$a = \arg\max_{a'} F_{a'[i,j]}, \qquad F_{a[i,j]} > Th_2 \tag{7}$$

where $Th_2 \in [0, 1]$ denotes a threshold that decides to execute a planning of the specified (j, a, i). Note that $F_{a[i,j]}$ denotes the transition probability from the feature index j to i by the action a.

3. Composition of the Feature Space and the State Space Dynamics Models

As mentioned in section 1, the dynamics model of the feature space is accurate but has a poor generalization, while the dynamics model of the state space has a good generalization. Thus, we develop a composite dynamics model of the feature space dynamics and the state space dynamics to exploit the both advantages. We call this model *MixFS dynamics model*.

3. 1 MixFS Dynamics Model

To use a dynamics model in the prioritized sweeping, a transition probability from the feature index i to j by the action a should be calculated. Hence, we define the MixFS dynamics model as

$$\phi' \approx \tilde{f}_a(\phi(x), x) \tag{8}$$

where $\tilde{f}_a$ denotes a general function approximator that estimates the succeeding feature vector ϕ' from the state x by the action a. This model is learned during the Dyna. Then we compute the feature transition matrix; for all feature index pairs $i, j \in \{1, \ldots, |\mathcal{K}|\}$ and action $a \in \mathcal{A}$,

$$F_{a[j,i]} \leftarrow \tilde{f}_a(\mathbf{e}_i, \mu_i)_{[j]} \tag{9}$$

where $\mathbf{e}_i \in \mathbb{R}^{|\mathcal{K}|\times 1}$ denotes a vector whose j-th element is 1 and the rest are 0. The obtained feature transition matrix is directly used in the prioritized sweeping part of the Dyna algorithm.

As the function approximator $\tilde{f}_a$, we choose simple linear models for two dynamics models, and combine them linearly. Thus, $\tilde{f}_a$ is defined by

$$\tilde{f}_a(\phi(x), x) = \delta F_a \phi(x) + \phi(x + \delta\tilde{x}_a(x)) \tag{10}$$

$$\delta\tilde{x}_a(x) = W_a \phi(x) + d_a \tag{11}$$

where $\delta F_a \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{K}|}$, $W_a \in \mathbb{R}^{\dim(\mathcal{X})\times|\mathcal{K}|}$, $d_a \in \mathbb{R}^{\dim(\mathcal{X})\times 1}$ are model parameters. For further understanding, let us think about a model, $\tilde{f}'_a(\phi(x), x) = \phi(x + \delta\tilde{x}_a(x))$. Here, $x + \delta\tilde{x}_a(x)$ is a linear state space dynamics model, that is, it can estimate the succeeding state from the state x by the action a. Thus, the model $\tilde{f}'_a(\phi(x), x)$ estimates the succeeding feature vector from x by a. The MixFS dynamics model eq. (10) is now clearly understood as a composite dynamics model.

3. 2 Learning MixFS Dynamics Model

Next, we derive an on-line learning algorithm for the model parameters of the MixFS dynamics model, with which the *Update models* part of the Dyna algorithm is replaced. Specifically, the learning algorithm updates δF_a, W_a, d_a so that $\tilde{f}_a(\phi(x), x)$ can estimate the succeeding feature vector ϕ' from the observation data x_n, ϕ_n, x_{n+1}, ϕ_{n+1}. Simply, we use an on-line gradient descent algorithm [11].

However, it is difficult to update W_a and d_a in a straightforward way, since these parameters are enveloped by the basis functions $\phi(x)$ that makes it complex to calculate the gradient. To overcome this difficulty, we separate the learning problem into two steps. First, the model parameters of the state space dynamics model W_a, d_a are updated so that $x + \delta\tilde{x}_a(x)$ can estimate the succeeding state x'. Second, the model parameter of the feature space dynamics model δF_a is updated so that $\tilde{f}_a(\phi(x), x)$ can estimate the succeeding feature vector ϕ'.

Thus, the model parameters of MixFS dynamics model are updated through the following algorithm:

```
Algorithm 2: Update MixFS dynamics model parameters
Input: current and succeeding state $x, x' \in \mathcal{X}$, action $a \in \mathcal{A}$,
       basis functions $\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{K}|\times 1}$, step size $\alpha$,
       current model parameters $\delta F_a$, $W_a$, $d_a$
Output: updated model parameters $\delta F'_a$, $W'_a$, $d'_a$
1: $\tilde{x}' \leftarrow x + W_a \phi(x) + d_a$
2: $\tilde{\phi}' \leftarrow \delta F_a \phi(x) + \phi(\tilde{x}')$
3: $W'_a \leftarrow W_a + \alpha(x' - \tilde{x}')\phi(x)^\top$
4: $d'_a \leftarrow d_a + \alpha(x' - \tilde{x}')$
5: $\delta F'_a \leftarrow \delta F_a + \alpha(\phi(x') - \tilde{\phi}')\phi(x)^\top$
```

3. 3 Computational Techniques

3. 3. 1 Fast Computation

After updating the model parameters of the MixFS, the feature transition matrix F_a should be computed by eq. (9), but it takes some computational cost. Since the feature transition matrix is required only when the planning is executed in the Dyna-MG, i.e. $PQueue \neq \emptyset$, the feature transition matrix have to be calculated only when it is demanded. To do this, $flag_a \in \{\texttt{true}, \texttt{false}\}$ is prepared for each action $a \in \mathcal{A}$ to judge whether the F_a is already calculated for the latest model parameters or not.

3. 3. 2 Constraint for Numerical Stability

The feature transition matrix F_a is assumed to be encoding transition probabilities of the MDP. However, due to the estimation error, F_a sometimes takes an irregular value, which makes the planning unstable. Thus, we constrain the F_a as follows:

```
for all $i, j \in \{1, \ldots, |\mathcal{K}|\}$:
    if $F_{a[j,i]} < 0$ then: $F_{a[j,i]} \leftarrow 0$
    if $F_{a[j,i]} > 1$ then: $F_{a[j,i]} \leftarrow 1$
for all $i \in \{1, \ldots, |\mathcal{K}|\}$:
    if $\sum_{j'} F_{a[j',i]} > 1$ then:
        for all $j \in \{1, \ldots, |\mathcal{K}|\}$: $F_{a[j,i]} \leftarrow \frac{F_{a[j,i]}}{\sum_{j'} F_{a[j',i]}}$
```

The first part constrains the range of the $F_{a[j,i]}$ to $[0, 1]$ since it represents a transition probability in the MDP. The second part constrains the sum of the transition probability w.r.t. the succeeding feature index j' from i by an action a, which should be 1.

4. Experimental Comparison of Dynamics Models

First, we compare the approximation accuracy of the two dynamics models in an environment of simple dynamics. One model is a traditional linear dynamics model of the feature space used in the Dyna-MG mentioned in section 2.2, which we refer as the *Simple dynamics model*. And the other is the proposed MixFS dynamics model.

We employ a robot that has 1-dimensional state space $\mathcal{X} \subset \mathbb{R}$ and only 1-element action set $\mathcal{A} = \{a\}$. The dynamics of the robot is defined as

$$x' = f_a(x) = \max(\min(1.2x + 0.5, 2.0), -1) \qquad (12)$$

where x' denotes the succeeding state from the state x by the action a. The experiment is performed as follows: (1) repeat N_{smpl} times: {generate x from uniform random distribution $[-2.5, 2.5]$, compute $x' = f_a(x)$, and train the models in an on-line manner}, (2) evaluate the RMS with $(x, \phi') \in \{(x_n, \phi'_n) | x_n{=}-2.5, -2.48, \ldots, 2.5; \phi'_n{=}\phi(f_a(x_n))\}$. We allocated 5 NGnet in $\mathcal{X}$ whose parameters are $\{(\mu_k, \Sigma_k) | \mu_k{=}-2, -1, 0, 1, 2; \Sigma_k{=}\frac{1}{9}\}$. We set $\alpha = 0.1$ as the step size parameter, and tested N_{smpl} in $\{10, 20, \ldots, 2000\}$.

Fig. 2 shows the approximation accuracy of each dynamics models (the mean of the RMS over 10 runs is plotted per N_{smpl}). The MixFS dynamics model has higher accuracy both in small N_{smpl} and large N_{smpl} than the Simple dynamics model. The possible reason is as follows. The dynamics of the robot is nearly linear in many regions of the state space, so the linear component of the state space dynamics of the MixFS, d_a, leads to good generalization from small samples. On the other hand, the composition of two dynamics models enhances the approximation capability, thus the MixFS dynamics model also obtains a higher accuracy from a large number of samples than the Simple one.

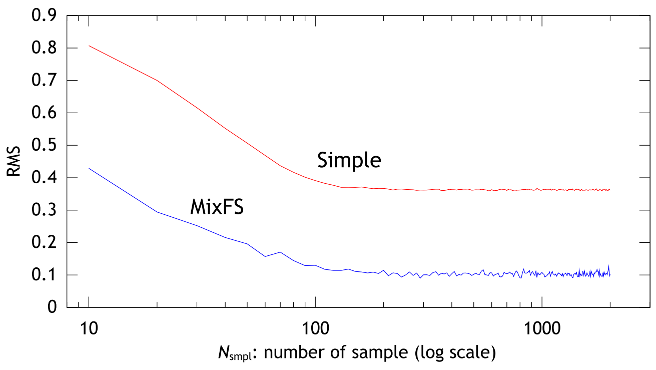

Fig. 2 Estimation errors of the two dynamics models, MixFS and Simple, per number of samples N_{smpl} in 1-dimensional environment. The MixFS dynamics model always shows better estimation accuracy than the Simple dynamics model.

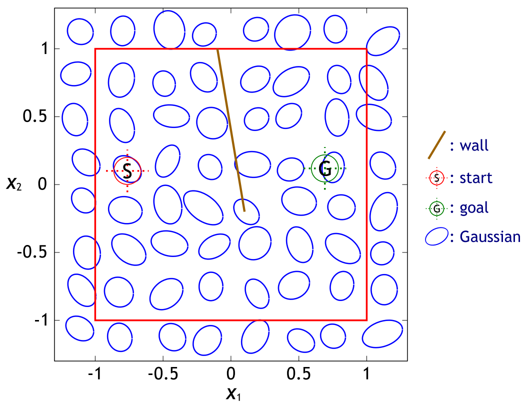

Fig. 3 The map of the robot navigation task. The state of the robot has 2-dimensional continuous value, $(x_1, x_2) \in \mathcal{X}_{\text{pl}}$, and the possible actions are $\mathcal{A} = \{up, left, down, right\}$.

5. Experimental Comparison of Dyna Performance

Next, we evaluate the dynamics models with the Dyna-MG algorithm in a robot navigation task on a 2-dimensional plane.

5.1 Experimental Setup

We employ an omniwheel mobile robot that can move in any direction on a 2-dimensional plane $(x_1, x_2) \in \mathcal{X}_{\text{pl}} = \{(x_1, x_2) | x_1{\in}[-1, 1], x_2{\in}[-1, 1]\}$ (Fig. 3). The state of the robot can be expressed as $x = (x_1, x_2)^\top$, and the possible actions are $\mathcal{A} = \{up, left, down, right\}$. There are *walls* which the robot can not cross.

The objective of the navigation task is to acquire a path with which the robot can move from the *start* to the *goal* shown in Fig. 3. If the succeeding state x' is at the *goal*, 1 is given as the reward; if $x' \notin \mathcal{X}_{\text{pl}}$, -0.5 is given; -0.003 is given for each action. Each episode begins with the *start* state $x(0) = x_s$, and ends if the robot has reached the *goal*, gone outside ($x' \notin \mathcal{X}_{\text{pl}}$), or $t > 100$.

We use NGnet with 64 BFs allocated as shown in Fig. 3 to

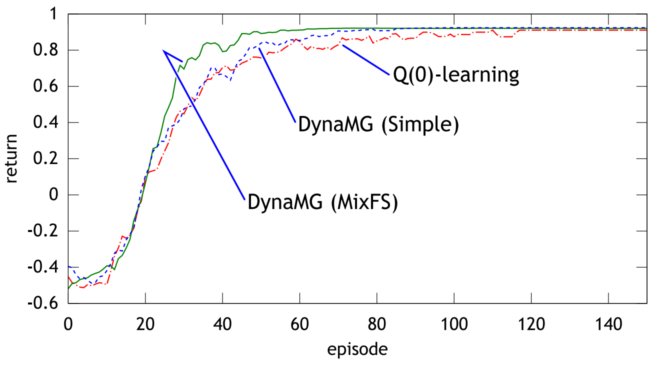

Fig. 4 Resulted learning curves of the robot navigation task. Each curve shows the mean of the return per episode over 25 runs. The `DynaMG (Simple)` denotes the Dyna-MG with the Simple dynamics model, and The `DynaMG (MixFS)` denotes the Dyna-MG with the MixFS dynamics model.

approximate the action value function. These BFs are allocated on a 8×8 grid with added random noise to each center and covariance.

The configuration of the Dyna-MG is $\gamma = 0.9$, $\alpha = 0.1$, $N_{\text{pln}} = 5$, $Th_1 = 0.2$, $Th_1 = 0.3$. For exploring actions, we use the Boltzmann policy selection with the temperature $\tau = 0.1$. These parameters and coefficients are chosen through preliminary experiments. We compared the Q(0)-learning (let $N_{\text{pln}} = 0$ in the Dyna-MG algorithm), the Dyna-MG with the MixFS dynamics model, and the Dyna-MG with the Simple dynamics model (same as Algorithm 1).

5.2 Result of Dyna-MG

Fig. 4 shows the learning curves (the mean of the return over 25 runs is plotted per episode) resulted from the three algorithms. The Dyna-MG with the MixFS is the fastest among the three. The Dyna-MG with the Simple dynamics model is slightly faster than the Q(0)-learning. This result is considered to be an effect of the planning. Moreover, the MixFS dynamics model can obtain a more accurate estimation than the Simple one even from a small number of samples. Thus, the planning in the Dyna-MG becomes more precise which makes the learning faster.

5.3 Embedding Reward Sources

If the state and the reward of the goal is known, we can embed it into the reward model as a prior knowledge. Dyna can exploit such kind of information by planning. A general way to embed such kind of information, i.e. the *reward sources*, into the reward model is formulated and solved as follows:

For given reward sources $\{(x_i, R_i) | i = 1, \dots, N_{\text{rsrc}}\}$, estimate $b_a, a \in \mathcal{A}$ to satisfies

$$\begin{bmatrix} R_1 \\ \vdots \\ R_{N_{\text{rsrc}}} \end{bmatrix} = \begin{bmatrix} \phi(x_1) & \cdots & \phi(x_{N_{\text{rsrc}}}) \end{bmatrix}^\top b_a. \quad (13)$$

The solution is

$$b_a = \begin{bmatrix} \phi(x_1) & \cdots & \phi(x_{N_{\text{rsrc}}}) \end{bmatrix}^{\top\sharp} \begin{bmatrix} R_1 \\ \vdots \\ R_{N_{\text{rsrc}}} \end{bmatrix} \quad (14)$$

where $\square^\sharp$ denotes the pseudo-inverse of the matrix $\square$.

Note that the reward model obtained by this method is slightly different from the correct one since the goal reward is given when the *succeeding* state is the goal state, but the above reward model says that the goal reward is given for any action from the goal state. The agent can not resolve this problem since initially it does not know about the dynamics. But, this problem is addressed through learning.

Again in the same navigation task, we compared the three algorithms, Q(0)-learning, the Dyna-MG with the MixFS dynamics model, and the Dyna-MG with the Simple dynamics model. The Q(0)-learning is exactly the same as the previous one. In the other two methods, the reward model is initialized by the above method where the goal state and the goal reward are given as reward sources.

Fig. 5 shows the resulted learning curves (the mean of the return over 25 runs is plotted per episode). Again, the Dyna-MG with the MixFS dynamics model is the fastest, and the Dyna-MG with the Simple dynamics model is faster than the Q(0)-learning. Here, it is natural that the fastest is the Dyna-MG with the MixFS dynamics model which is more precise than the Simple one. Moreover, these two Dyna-MGs are faster than that in the previous experiment (see Fig. 4). This is possible because the planning of the Dyna-MG can utilize the reward sources information embedded into the reward model by the above method, which makes the Dyna-MG faster.

6. Conclusions

In this paper, we develop a composite dynamics model of the feature space dynamics and the state space dynamics, which is named MixFS dynamics model. The MixFS can exploit the advantages of both dynamics models, that is, the generalization of the state space dynamics model and the accuracy of the feature space dynamics model. We contribute to combine the two dynamics models and make it work with the Dyna algorithm, and to derive an on-line learning method of the developed model.

The simulation result demonstrates that the MixFS can exploit the advantages of both dynamics models as we have expected. Concretely, the MixFS can estimate the dynamics more precisely both from a small number and large number

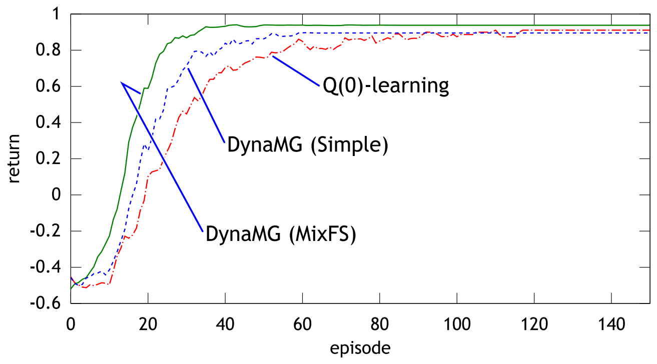

Fig. 5 Resulted learning curves of the robot navigation task. Each curve shows the mean of the return per episode over 25 runs. The `DynaMG (Simple)` denotes the Dyna-MG with the Simple dynamics model, and The `DynaMG (MixFS)` denotes the Dyna-MG with the MixFS dynamics model. The reward models of these Dyna-MG algorithms are initialized so that they approximate the goal reward. The Q(0)-learning is exactly the same as the Fig. 4.

of samples than the conventional linear dynamics model of a feature space. As a result, the Dyna-MG with the MixFS is faster than that with the conventional dynamics model.

Bibliography

[1] R.S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," In Proceedings of the Seventh International Conference on Machine Learning, pp.216–224, Morgan Kaufmann, 1990.

[2] 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.

[3] A. Rottmann and W. Burgard, "Adaptive autonomous control using online value iteration with gaussian processes," the IEEE Internactional Conference in Robotics and Automation (ICRA'09), pp.2106–2111, 2009.

[4] A.M. Farahmand, A. Shademan, M. Jägersand, and C. Szepesvári, "Model-based and model-free reinforcement learning for visual servoing," the IEEE Internactional Conference in Robotics and Automation (ICRA'09), pp.2917–2924, Kobe, Japan, May 2009.

[5] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," International Journal of Control, Automation, and Systems, vol.5, no.6, pp.674–680, 2007.

[6] C.J.C.H. Watkins, "Learning from delayed rewards," PhD thesis, Cambridge University, 1989.

[7] G.A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994. citeseer.ist.psu.edu/rummery94line.html

[8] T. Kondo and K. Ito, "A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control," Robotics and Autonomous Systems, vol.46, no.2, pp.111–124, 2004.

[9] J. Morimoto and K. Doya, "Reinforcement learning of dynamic motor sequence: Learning to stand up," the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), pp.1721–1726, 1998.

[10] 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.

[11] C.M. Bishop, Pattern recognition and machine learning, Springer, 2006.