

Multiple Model-based Reinforcement Learning

Kenji Doya^{*1235}, Kazuyuki Samejima³,
Ken-ichi Katagiri⁴⁵, and Mitsuo Kawato³⁴⁵

*doya@isd.atr.co.jp

¹Information Sciences Division, ATR International
2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan
phone: +81-774-95-1251; fax: +81-774-95-1259

²CREST, Japan Science and Technology Corporation

³Kawato Dynamic Brain Project, ERATO
Japan Science and Technology Corporation

⁴ATR Human Information Processing Research Laboratories

⁵Nara Institute of Science and Technology

June 1, 2000

Abstract

We propose a modular reinforcement learning architecture for non-linear, non-stationary control tasks. The basic idea is to decompose a complex task into multiple domains in space and time based on the predictability of the environmental dynamics. The system is composed of multiple modules, each of which consists of a state prediction model and a reinforcement learning controller. The “responsibility signal,” which is given by the softmax function of the prediction errors, is used to weight the outputs of multiple modules as well as to gate the learning of the prediction models and the reinforcement learning controllers. The performance of the architecture, which we call multiple model-based reinforcement learning (MMRL), is demonstrated in a non-linear, non-stationary control task of swinging up a pendulum with variable physical parameters.

1 Introduction

The application of reinforcement learning (RL) to real-world control problems has been limited by the facts that learning is very slow for highly non-linear problems and that convergence is not guaranteed for non-stationary environments. These problems have motivated the introduction of modular or hierarchical RL architectures (Singh 1992; Dayan and Hinton 1993; Littman et al. 1995; Wiering and Schmidhuber 1998; Parr and Russel 1998; Sutton et al. 1999). The basic problem in modular or hierarchical RL is how to decompose a complex task into different subtasks.

This paper presents a new RL architecture based on multiple modules, each of which is composed of a state prediction model and a RL controller. With this architecture, a non-linear and/or non-stationary control task is decomposed in space and time based on the local predictability of the environmental dynamics.

The “mixture of experts” architecture (Jacobs et al. 1991) has previously been applied to non-linear or non-stationary control tasks (Gomi and Kawato 1993; Cacciatore and Nowlan 1994). However, the success of such modular architecture depends strongly on the capability of the gating network to decide which of the given modules should be recruited at any particular moment.

An alternative approach is to provide each of the experts with a prediction model of the environment and to utilize the prediction errors for the selection of the controllers. In Narendra et al. (1995), the model that makes the smallest prediction error among a fixed set of prediction models is selected, and its associated single controller is used for control. However, when the prediction models are to be trained with little prior knowledge, task decomposition is initially far from optimal. Thus the use of ‘hard’ competition can lead to sub-optimal task decomposition.

Based on the Bayesian statistical framework, Pawelzik et al. (1996) proposed the use of annealing in a ‘soft’ competition network for time series prediction and segmentation. A similar mechanism is used by Tani and Nolfi (1999) for hierarchical sequence prediction. The use of the softmax function for module selection and combination was recently proposed for a tracking control paradigm as the “Multiple Paired Forward-Inverse Models (MPFIM)” (Wolpert and Kawato 1998; Wolpert et al. 1998; Haruno et al. 1999).

In this paper, we extend the idea of a softmax combination of modules to the paradigm of reinforcement learning. The resulting learning architecture, which we call “Multiple Model-based Reinforcement Learning (MMRL),” learns to decompose a non-linear and/or non-stationary task through the competition and cooperation of multiple prediction models and reinforcement learning controllers.

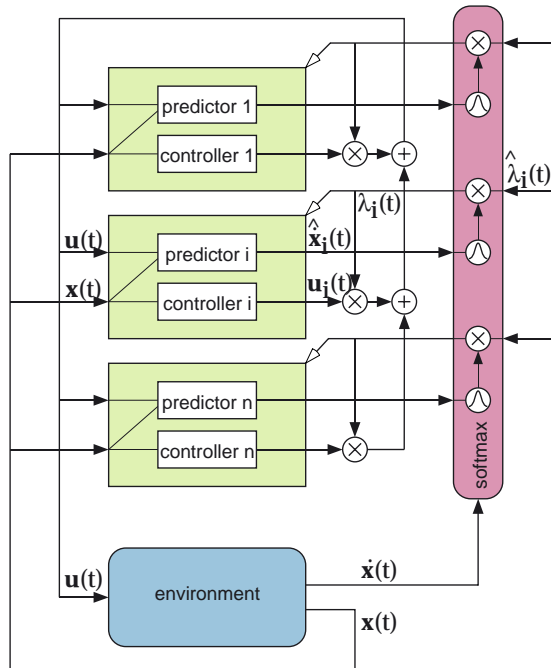


Figure 1: Schematic diagram of multiple model-based reinforcement learning (MMRL) architecture.

In the following sections, we first formulate the basic MMRL architecture (Section 2) and then describe its implementation in discrete-time and continuous-time cases, including “Multiple Linear Quadratic Controllers (MLQC)” (Section 3). We test the performance of the MMRL architecture in a non-linear, non-stationary control task of swinging up a pendulum with variable physical parameters (Section 4). It is shown that available modules are flexibly allocated for different domains in space and time based on the task demands.

2 Multiple Model-based Reinforcement Learning

Figure 1 shows the overall organization of the multiple model-based reinforcement learning (MMRL) architecture. It is composed of n modules, each of which consists of a state prediction model and a reinforcement learning controller.

The basic idea of this modular architecture is to decompose a non-linear and/or non-stationary task into multiple domains in space and time so that within each of the domains the environmental dynamics is well predictable. The action output of the RL controllers as well as the learning rates of both the predictors and the controllers are weighted by the

“responsibility signal,” which is a Gaussian softmax function of the errors in the outputs of the prediction models. The advantage of this module selection mechanism is that the areas of specialization of the modules are determined in a bottom-up fashion based on the nature of the environment. Furthermore, for each area of module specialization, the design of the control strategy is facilitated by the availability of the local model of the environmental dynamics.

In the following, we consider a discrete-time environment

$$\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t)) \quad (1)$$

and a continuous-time environment

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)). \quad (2)$$

The goal of reinforcement learning (Sutton and Barto 1998) is to find the policy

$$\mathbf{u}(t) = g(\mathbf{x}(t)) \quad (3)$$

that maximizes the cumulative future reward

$$E \left[\sum_{k=0}^{\infty} \gamma^k r(t+k) \right] \quad (4)$$

or

$$E \left[\int_t^{\infty} e^{-\frac{s-t}{\tau}} r(s) ds \right], \quad (5)$$

where $0 \leq \gamma \leq 1$ and $0 < \tau$ are the parameters for discounting future reward. The cumulative future reward as the function of the current state $\mathbf{x}(t)$ is defined as the state value function $V(\mathbf{x})$.

2.1 Responsibility Signal

The task for the prediction model in each module is to predict the next state $\mathbf{x}(t+1)$ (discrete-time) or the state change $\dot{\mathbf{x}}(t)$ (continuous-time). We denote the output of the i -th module as $\hat{\mathbf{x}}_i(t+1)$ and $\hat{\dot{\mathbf{x}}}_i(t)$ and the prediction error as

$$E_i(t) = \|\hat{\mathbf{x}}_i(t) - \mathbf{x}(t)\|^2 \quad (6)$$

for discrete-time and

$$E_i(t) = \|\hat{\dot{\mathbf{x}}}_i(t) - \dot{\mathbf{x}}(t)\|^2 \quad (7)$$

for continuous-time.

The responsibility signal $\lambda_i(t)$ (Wolpert and Kawato 1998; Haruno et al. 1999) for the i -th module is then given by the softmax function of prediction errors

$$\lambda_i(t) = \frac{e^{-\frac{E_i(t)}{2\sigma^2}}}{\sum_{j=1}^n e^{-\frac{E_j(t)}{2\sigma^2}}}, \quad (8)$$

where σ is a parameter that controls the sharpness of module selection. In the Bayesian framework, this responsibility signal $\lambda_i(t)$ is interpreted as the posterior probability that the i -th model generated the data $\mathbf{x}(t)$ or $\dot{\mathbf{x}}(t)$ under a Gaussian noise assumption (Pawelzik et al. 1996).

2.2 Module Weighting by Responsibility Signal

In the MMRL architecture, the responsibility signal $\lambda_i(t)$ is used for four purposes: 1) weighting the state prediction outputs; 2) gating the learning of prediction models; 3) weighting the action outputs; and 4) gating the learning of reinforcement learning controller.

1) State prediction: The outputs of the prediction models are linearly weighted by the responsibility signal $\lambda_i(t)$ to make a prediction of the next state

$$\hat{\mathbf{x}}(t+1) = \sum_{i=1}^n \lambda_i(t) \hat{\mathbf{x}}_i(t+1) \quad (9)$$

in discrete-time, and the vector field

$$\hat{\dot{\mathbf{x}}}(t) = \sum_{i=1}^n \lambda_i(t) \hat{\dot{\mathbf{x}}}_i(t) \quad (10)$$

in continuous-time. These predictions are used in model-based RL algorithms and also for the annealing of σ as described later.

2) Prediction model learning: $\lambda_i(t)$ is then used for weighting the parameter update of the prediction models. Namely, the error signal for the i -th prediction model is given by

$$\lambda_i(t)(\hat{\mathbf{x}}_i(t+1) - \mathbf{x}(t+1)) \quad (11)$$

for discrete-time and

$$\lambda_i(t)(\hat{\dot{\mathbf{x}}}_i(t) - \dot{\mathbf{x}}(t)) \quad (12)$$

for continuous-time.

3) Action output: The outputs of reinforcement learning controllers

$$\mathbf{u}_i(t) = g_i(\mathbf{x}(t)) \quad (13)$$

are linearly weighted by $\lambda_i(t)$ to make the action output to the environment

$$\mathbf{u}(t) = \sum_{i=1}^n \lambda_i(t) \mathbf{u}_i(t). \quad (14)$$

4) Reinforcement learning: $\lambda_i(t)$ is also used for weighting the learning of the RL controllers. The actual equation for the parameter update varies with the choice of the RL algorithms, which are detailed in the next section. When a temporal difference (TD) algorithm (Barto et al. 1983; Sutton 1988; Doya 2000) is used, the TD error $\delta(t)$ weighted by the responsibility signal

$$\delta_i(t) = \lambda_i(t)\delta(t) \quad (15)$$

is used as the teaching signal for the i -th RL controller.

Using the same weighting factor $\lambda_i(t)$ for training the prediction models and the RL controllers helps each RL controller to learn an appropriate policy and its value function for the context under which its paired prediction model makes valid predictions.

2.3 Responsibility predictors

If there is some knowledge or belief about module selection, it can be represented as the “responsibility prediction signal” $\hat{\lambda}_i(t)$ (Wolpert and Kawato 1998; Haruno et al. 1999) and incorporated in responsibility estimation as

$$\lambda_i(t) = \frac{\hat{\lambda}_i(t)e^{-\frac{E_i(t)}{2\sigma^2}}}{\sum_{j=1}^n \hat{\lambda}_j(t)e^{-\frac{E_j(t)}{2\sigma^2}}}. \quad (16)$$

If the responsibility signals $\hat{\lambda}_i(t)$ ($i = 1, \dots, n$) sum up to one, then they can be interpreted as the prior probability $p(i)$ for selecting the i -th model.

Here, we consider two special cases of responsibility predictors: one based on spatial locality and the other based on temporal continuity.

2.3.1 Spatial locality

We consider a Gaussian spatial prior

$$\hat{\lambda}_i(t) = \frac{e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c}_i)'M_i^{-1}(\mathbf{x}-\mathbf{c}_i)}}{\sum_{j=1}^n e^{-\frac{1}{2}(\mathbf{x}-\mathbf{c}_j)'M_j^{-1}(\mathbf{x}-\mathbf{c}_j)}}, \quad (17)$$

where \mathbf{c}_i is the center of the area of specialization and M_i is a covariance matrix that specifies the shape.

These parameters are updated according to the history of module activation as follows.

$$\mathbf{c}_i(t+1) = \mathbf{c}_i(t) + \eta_c \lambda_i(t)(\mathbf{x}(t) - \mathbf{c}_i(t)), \quad (18)$$

$$M_i(t+1) = M_i(t) + \eta_M \lambda_i(t)[-M_i(t) + \mathbf{x}(t) - \mathbf{c}_i(t)](\mathbf{x}(t) - \mathbf{c}_i(t))', \quad (19)$$

where η_c and η_M are update rates.

2.3.2 Temporal continuity

The prior knowledge that module switching is not very frequent can be incorporated by setting the responsibility prediction signal based on the previous responsibility signal. We consider in discrete-time

$$\hat{\lambda}_i(t) = \frac{\lambda_i(t-1)^\alpha}{\sum_j \lambda_j(t-1)^\alpha}, \quad (20)$$

where $0 < \alpha < 1$ is a parameter that controls the strength of the memory effect. From the recursive expression of the responsibility signal

$$\lambda_i(t) = \frac{\lambda_i(t-1)^\alpha e^{-\frac{E_i(t)}{2\sigma^2}}}{Z_i(t)}, \quad (21)$$

where $Z_i(t)$ is a normalizing constant, the responsibility signal is given by

$$\lambda_i(t) = \frac{\prod_{k=0}^t e^{-\alpha^k \frac{E_i(t-k)}{2\sigma^2}}}{Z_i(t)} = \frac{e^{-\frac{1}{2\sigma^2} \sum_{k=0}^t \alpha^k E_i(t-k)}}{Z_i(t)}. \quad (22)$$

This can be equivalently implemented by using a short-term average of the prediction error

$$E_i(t) = \sum_{k=0}^t \alpha^k \|\hat{\mathbf{x}}_i(t-k) - \mathbf{x}(t-k)\|^2 \quad (23)$$

instead of the instantaneous prediction error (6) in calculating the responsibility signal by (8) or (16). The above short-term average error can be recursively obtained by

$$E_i(t) = \alpha E_i(t-1) + \|\hat{\mathbf{x}}_i(t) - \mathbf{x}(t)\|^2. \quad (24)$$

In continuous-time, temporal continuity of the responsibility signal is also implemented by using a short-term average prediction error

$$\tau_p \frac{d}{dt} E_i(t) = -E_i(t) + \|\hat{\dot{\mathbf{x}}}_i(t) - \dot{\mathbf{x}}(t)\|^2 \quad (25)$$

instead of (7). The use of these low-pass filtered prediction errors for responsibility prediction is helpful in avoiding chattering of the responsibility signal (Pawelzik et al. 1996).

3 Implementation of MMRL Architecture

For the RL controllers, it is generally possible to use model-free RL algorithms, such as actor-critic and Q-learning. However, because the prediction models of the environmental dynamics are intrinsic components of the architecture, it is advantageous to utilize these prediction models not just for module selection but also for designing RL controllers. In the following, we describe the use of model-based RL algorithms for discrete-time and continuous-time cases. One special implementation for continuous-time is the use of multiple ‘‘linear quadratic controllers’’ derived from linear dynamic models and quadratic reward models.

3.1 Discrete-time MMRL

Now we consider implementation of the MMRL architecture for discrete-time. The standard way of utilizing a predictive model in RL is to use it for action selection by the one-step search

$$\mathbf{u}(t) = \arg \max_{\mathbf{u}} [\hat{r}(\mathbf{x}(t), \mathbf{u}) + \gamma V(\hat{\mathbf{x}}(t+1))], \quad (26)$$

where $\hat{r}(\mathbf{x}(t), \mathbf{u})$ is a reward prediction model and $V(\hat{\mathbf{x}}(t+1))$ is the value function for the state $\hat{\mathbf{x}}(t+1)$, which is predicted from state $\mathbf{x}(t)$ and an action \mathbf{u} .

In order to implement this algorithm, we provide each module with a dynamic model $f_i(\mathbf{x}, \mathbf{u})$, a reward model $\hat{r}_i(\mathbf{x}, \mathbf{u})$, and a value function $V_i(\mathbf{x})$. The outputs of the dynamic models

$$\hat{\mathbf{x}}_i(t) = f_i(\mathbf{x}(t-1), \mathbf{u}(t-1)) \quad (27)$$

are compared with the actually observed state $\mathbf{x}(t)$ to calculate the responsibility signal $\lambda_i(t)$ according to (8). The responsibility signal is then used to weight the reward prediction

$$\hat{r}(\mathbf{x}(t), \mathbf{u}) = \sum_{i=1}^n \lambda_i(t) \hat{r}_i(\mathbf{x}(t), \mathbf{u}) \quad (28)$$

and the value function

$$V(\hat{\mathbf{x}}(t+1)) = \sum_{i=1}^n \lambda_i(t) V_i(f_i(\mathbf{x}(t), \mathbf{u})), \quad (29)$$

which are used in action selection by (26).

The parameters of these models are updated by the weighted error signals $\lambda_i(t)(\hat{\mathbf{x}}_i(t) - \mathbf{x}(t))$ for the dynamic model, $\lambda_i(t)(\hat{r}_i(\mathbf{x}(t), \mathbf{u}(t)) - r(t))$ for the reward model, and $\lambda_i(t)\delta(t)$ for the value model, where $\delta(t)$ is the the temporal difference error

$$\delta(t) = r(t) + \gamma V(\mathbf{x}(t+1)) - V(\mathbf{x}(t)). \quad (30)$$

3.2 Continuous-time MMRL

Next we consider a continuous-time MMRL architecture. A model-based RL algorithm for a continuous-time, continuous-state system (2) is derived from the Hamilton-Jacobi-Bellman (HJB) equation

$$\frac{1}{\tau} V(\mathbf{x}(t)) = \max_{\mathbf{u}} \left[r(\mathbf{x}(t), \mathbf{u}) + \frac{\partial V(\mathbf{x}(t))}{\partial \mathbf{x}} f(\mathbf{x}(t), \mathbf{u}) \right], \quad (31)$$

where τ is the time constant of reward discount (Doya 2000). Under the assumptions that the system is linear with respect to the action and the action cost is convex, a greedy policy is given by

$$\mathbf{u} = g \left(\frac{\partial f(\mathbf{x}, \mathbf{u})'}{\partial \mathbf{u}} \frac{\partial V(\mathbf{x})'}{\partial \mathbf{x}} \right), \quad (32)$$

where $\frac{\partial V(\mathbf{x})'}{\partial \mathbf{x}}$ is a vector representing the steepest ascent direction of the value function, $\frac{\partial f(\mathbf{x}, \mathbf{u})'}{\partial \mathbf{u}}$ is a matrix representing the input gain of the dynamics, and g is a sigmoid function whose shape is determined by the control cost (Doya 2000).

To implement the HJB based algorithm, we provide each module with a dynamic model $f_i(\mathbf{x}, \mathbf{u})$ and a value model $V_i(\mathbf{x})$. The outputs of the dynamic models

$$\hat{\mathbf{x}}_i(t) = f_i(\mathbf{x}(t), \mathbf{u}(t)) \quad (33)$$

are compared with the actually observed state dynamics $\dot{\mathbf{x}}(t)$ to calculate the responsibility signal $\lambda_i(t)$ according to (8).

The model outputs are linearly weighted by $\lambda_i(t)$ for state prediction

$$\hat{\mathbf{x}}(t) = \sum_{i=1}^n \lambda_i(t) f_i(\mathbf{x}(t), \mathbf{u}(t)), \quad (34)$$

and value function estimation

$$V(\mathbf{x}) = \sum_{i=1}^n \lambda_i(t) V_i(\mathbf{x}). \quad (35)$$

The derivatives of the dynamic models $\frac{\partial f_i(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}$ and value models $\frac{\partial V_i(\mathbf{x})}{\partial \mathbf{x}}$ are used to calculate policy (32) for each module

$$\mathbf{u}_i = g \left(\frac{\partial f_i(\mathbf{x}, \mathbf{u})'}{\partial \mathbf{u}} \frac{\partial V_i(\mathbf{x})'}{\partial \mathbf{x}} \right). \quad (36)$$

They are then weighted by $\lambda_i(t)$ according to (14) to make the actual action.

Learning is based on the weighted prediction errors $\lambda_i(t)(\hat{\mathbf{x}}_i(t) - \dot{\mathbf{x}}(t))$ for dynamic models and $\lambda_i(t)\delta(t)$ for value models, where $\delta(t)$ is the continuous-time temporal difference error (Doya 2000)

$$\delta(t) = \hat{r}(t) - \frac{1}{\tau} V(t) + \dot{V}(t). \quad (37)$$

3.3 Multiple Linear Quadratic Controllers

In a modular architecture like the MMRL, the use of universal non-linear function approximators with large degrees of freedom can be problematic because it can lead to an undesired solution in which a single module tries to handle most of the task domain. The use of linear models for the prediction models and the controllers is a reasonable choice because local linear models have been shown to have good properties of quick learning and good

generalization (Schaal and Atkeson 1996). Furthermore, if the reward function is locally approximated by a quadratic function, then we can use a *linear quadratic controller* (see, e.g., Bertsekas 1995) for the RL controller design.

We use a local linear dynamic model

$$\hat{\mathbf{x}}(t) = A_i(\mathbf{x}(t) - \mathbf{x}_i) + B_i\mathbf{u}(t) \quad (38)$$

and a local quadratic reward model

$$\hat{r}_i(\mathbf{x}(t), \mathbf{u}(t)) = -(\mathbf{x}(t) - \mathbf{x}_i)'Q_i(\mathbf{x}(t) - \mathbf{x}_i) - \mathbf{q}_i'\mathbf{x}(t) - q_i - \mathbf{u}(t)'R_i\mathbf{u}(t) \quad (39)$$

for each module, where \mathbf{x}_i is the center of local prediction.

The value function is given by the quadratic form

$$V_i(\mathbf{x}) = -(\mathbf{x} - \mathbf{p}_i)'P_i(\mathbf{x} - \mathbf{p}_i) - p_i. \quad (40)$$

The matrix P_i is given by solving the Riccati equation

$$0 = \frac{1}{\tau}P_i - P_iA_i - A_i'P_i + P_iB_iR_i^{-1}B_i'P_i - Q_i, \quad (41)$$

and the center \mathbf{p}_i of the value function is given by solving the linear equation

$$0 = \frac{1}{\tau}P_i\mathbf{p}_i - P_iA_i\mathbf{x}_i - A_i'P_i\mathbf{p}_i + P_iB_iR_i^{-1}B_i'P_i\mathbf{p}_i + \frac{1}{2}\mathbf{q}_i. \quad (42)$$

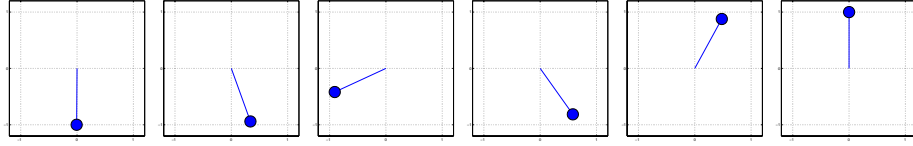
Then the optimal feedback control for each module is given by the linear feedback

$$\mathbf{u}_i(t) = -R_i^{-1}B_i'P_i(\mathbf{x} - \mathbf{p}_i). \quad (43)$$

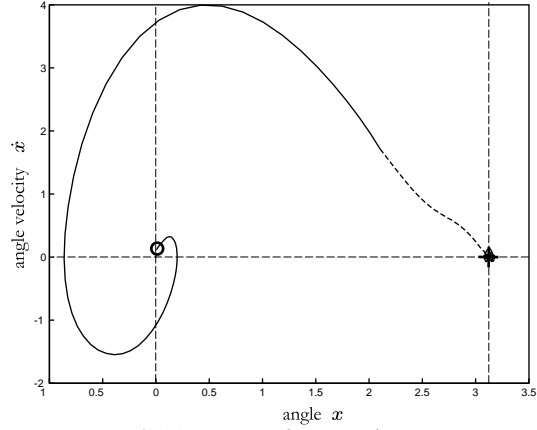
The action output is given by weighting these controller outputs by the responsibility signal $\lambda_i(t)$:

$$\mathbf{u}(t) = \sum_{i=1}^n \lambda_i(t)\mathbf{u}_i(t). \quad (44)$$

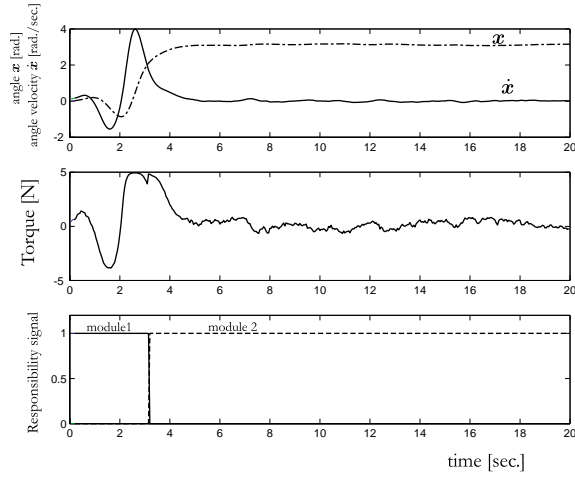
The parameters of the local linear models A_i , B_i , and \mathbf{x}_i and those of the quadratic reward models Q_i , \mathbf{q}_i , q_i , and R_i are updated by the weighted prediction errors $\lambda_i(t)(\hat{\mathbf{x}}_i(t) - \dot{\mathbf{x}}(t))$ and $\lambda_i(t)(r_i(\mathbf{x}, \mathbf{u}) - r(t))$, respectively. When we assume that the update of these models is slow enough, then the Riccati equations (41) may be recalculated only intermittently. We call this method ‘‘Multiple Linear Quadratic Controllers’’ (MLQC).



(a)



(b) Trajectory of system dynamics



(c) Time course of system dynamics and responsibility signal

Figure 2: (a) Example of swing-up performance. Dynamics are given by $ml^2\ddot{\theta} = -mgl \sin \theta - \mu\dot{\theta} + T$. Physical parameters are $m = l = 1$, $g = 9.8$, $\mu = 0.1$, and $T^{\max} = 5.0$. (b) Trajectory from the initial state $(0[\text{rad}], 0.1[\text{rad/s}])$. o : start, $+$: goal. Solid line: module 1. Dashed line: module 2. (c) Time course of the state (top), the action (middle), and the responsibility signal (bottom).

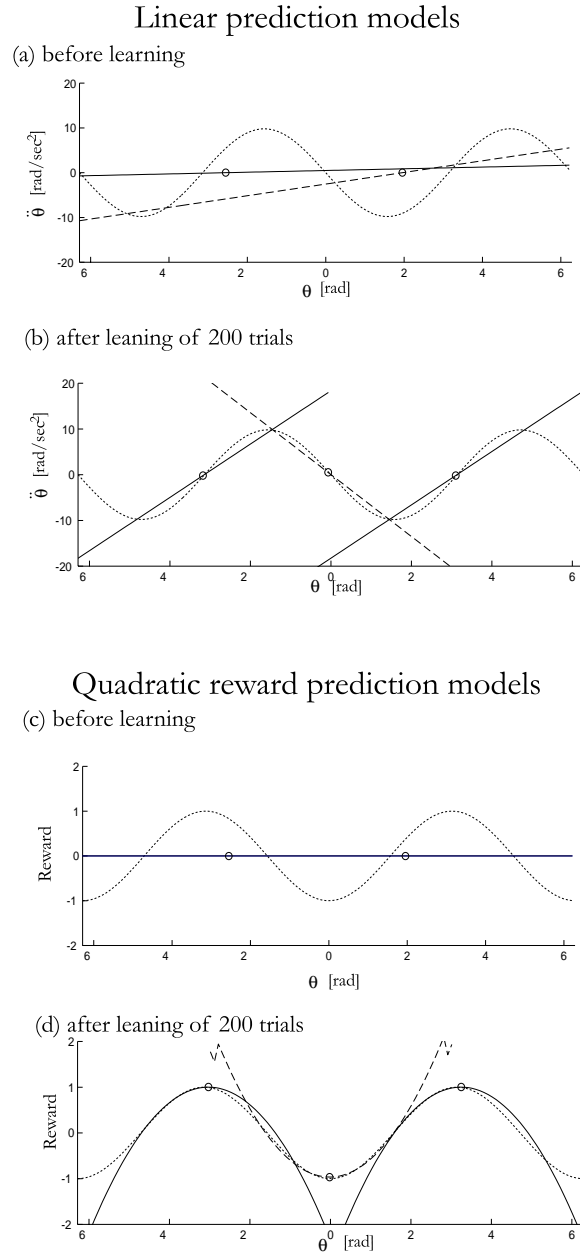


Figure 3: Development of state and reward prediction of models. (a) and (b): outputs of state prediction models before (a) and after (b) learning. (c) and (d): outputs of the reward prediction model before (c) and after (d) learning. Solid line: module 1. Dashed line: module 2; dotted line: targets (\ddot{x} and r). \circ : centers of spatial responsibility prediction \mathbf{c}_i .

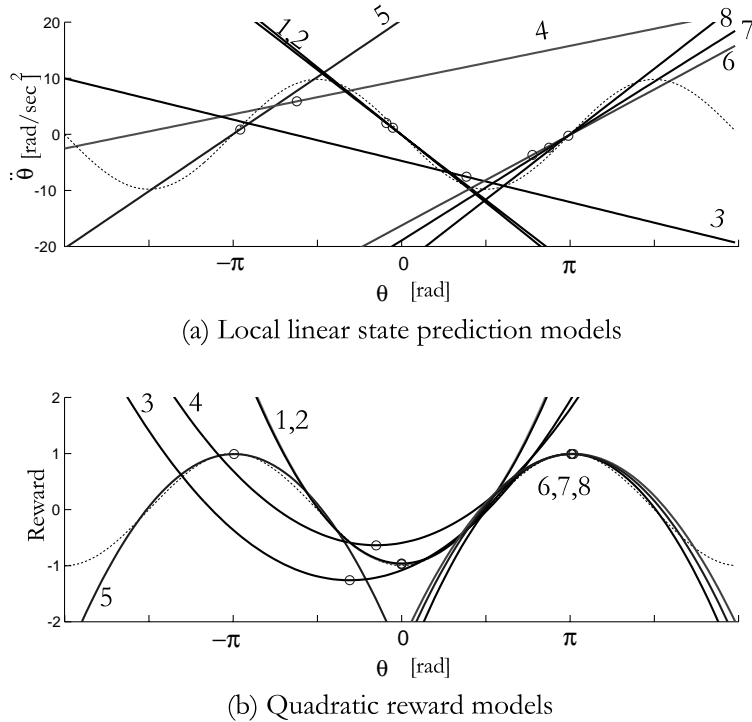


Figure 4: Outputs of 8 modules. (a) state prediction models. (b) reward prediction models.

4 Simulation: Pendulum Swing Up Task

In order to test the effectiveness of the MMRL architecture, we first applied the MLQC algorithm described in Section 3.3 to the task of swinging up a pendulum with limited torque (Figure 2) (Doya 2000). The driving torque T is limited in $[-T^{\max}, T^{\max}]$ with $T^{\max} < mgl$. The pendulum has to be swung back and forth at the bottom to build up enough momentum for a successful swing up.

The state space was 2-dimensional, i.e. $\mathbf{x} = (\theta, \dot{\theta})' \in S \times R$, and the action was $\mathbf{u} = T$. The reward was given by the height of the tip and the negative squared torque

$$r(\mathbf{x}, \mathbf{u}) = -\cos \theta - \frac{1}{2}RT^2. \quad (45)$$

We devised the following automatic annealing process for the parameter σ of the softmax function for the responsibility signal (8).

$$\sigma_{k+1} = \eta a E_k + (1 - \eta)\sigma_k, \quad (46)$$

where k denotes the number of trial and E_k is the average state prediction error during the k -th trial. The parameters were $\eta = 0.25$, $a = 2$, and the initial value set as $\sigma_0 = 4$.

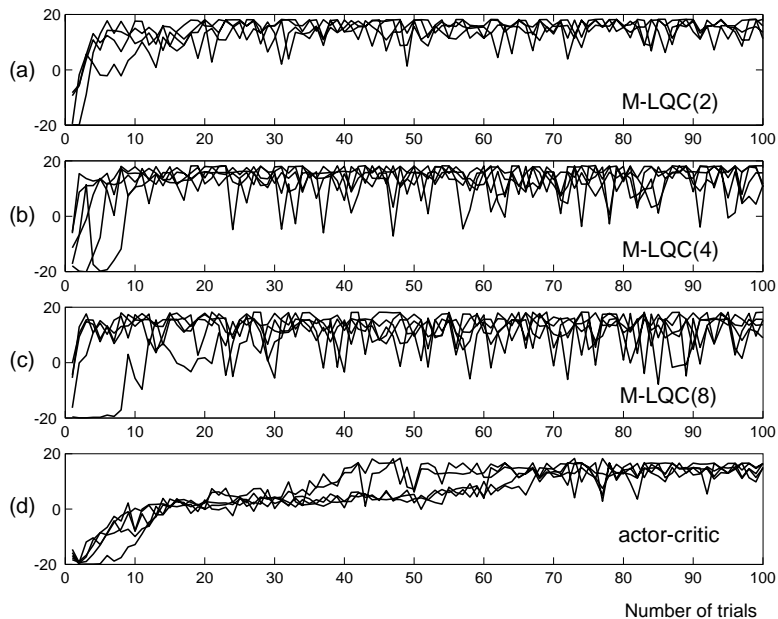


Figure 5: Learning curves for the pendulum swing up task. (a) $n=2$ modules. (b) 4 modules. (c) 8 modules. (d) non-modular architecture.

4.1 Task Decomposition in Space: Non-linear Control

We first used two modules, each of which had a linear dynamic model (38) and a quadratic reward model (39). The centers of the local linear dynamic models were initially placed randomly with the angular component in $[-\pi, \pi]$.

Each trial was started from a random position of the pendulum and lasted for 30 seconds.

Figure 2 shows an example of swing up performance from the bottom position. Initially, the first prediction model predicts the pendulum motion better than the second one, so the responsibility signal λ_1 becomes close to 1. Thus the output of the first RL controller u_1 , which destabilizes the bottom position, is used for control. As the pendulum is driven away from the bottom, the second prediction model predicts the movement better, so λ_2 becomes higher and the second RL controller takes over and stabilizes the upright position.

Figure 3 shows the changes of linear prediction models and quadratic reward models before and after learning. The two linear prediction models approximated the non-linear gravity term. The model centered at the bottom predicts the negative feedback acceleration, while the model centered at the top predicts the positive feedback acceleration. The two reward models also approximated the cosine reward function using parabolic curves.

Figure 4 shows the dynamic and reward models when there were eight modules. Two modules were specialized for the bottom position and three modules were specialized near the top position, while two other modules were centered somewhere in between. The result shows that proper modularization is possible even when there are redundant modules.

Figure 5 compares the time course of learning by MLQC with two, four, and eight modules and a non-modular actor-critic (Doya 2000). Learning was fastest with two modules, but the slowest performance of 8-module MLQC was still much faster than the non-modular architecture.

An interesting feature of the MLQC strategy is that qualitatively different controllers are derived by the solutions of the Riccati equations (41). The controller at the bottom is a positive feedback controller that de-stabilizes the equilibrium with a low reward, while the controller at the top is a typical linear quadratic regulator that stabilizes the upright state. Another important feature of the MLQC is that the modules were flexibly switched simply based on the prediction errors. Successful swing up was achieved without any top-down planning of the complex sequence.

4.2 Task Decomposition in Time: Non-stationary Pendulum

We then tested the effectiveness of the MMRL architecture for the both non-linear and non-stationary control task in which mass m and length l of the pendulum were changed every trial.

We used four modules, each of which had a linear dynamic model (38) and a quadratic reward model (39). The centers \mathbf{x}_i of the local linear prediction models were initially set randomly. Each trial was started from a random position with $\theta \in [-\pi/4, \pi/4]$ and lasted for 40 seconds.

We implemented responsibility prediction with $\tau_c = 50$, $\tau_M = 200$, and $\tau_p = 0.1$. The parameters of annealing were $\eta = 0.1$, $a = 2$, and an initial value of $\sigma_0 = 10$.

In the first 50 trials, the physical parameters were fixed at $\{m = 1.0, l = 1.0\}$. Figure 6(a) shows the change in the position gain ($\{A_{21}\} = \frac{\partial \ddot{\theta}}{\partial \theta}$) of the four prediction models. The control performance is shown in Figure 6(b). Figures 6(c,d,e) show prediction models in the section of $\{\theta = 0, T = 0\}$.

Initial position gains are set randomly (Figure 6(c)). After 50 trials, module 1 and module 2 both specialized in the bottom region ($\theta \simeq 0$) and learned similar prediction models. Module 3 and module 4 also learned the same prediction model in the top region ($\theta \simeq \pi$) (Figure 6(d)). Accordingly, the RL controllers in module 1 and module 2 learned a reward model with a minimum near $(0, 0)'$ and a destabilizing feedback policy was given by (15)-(17). Module 3 and module 4 also learned a reward model with a peak near $(\pi, 0)'$ and

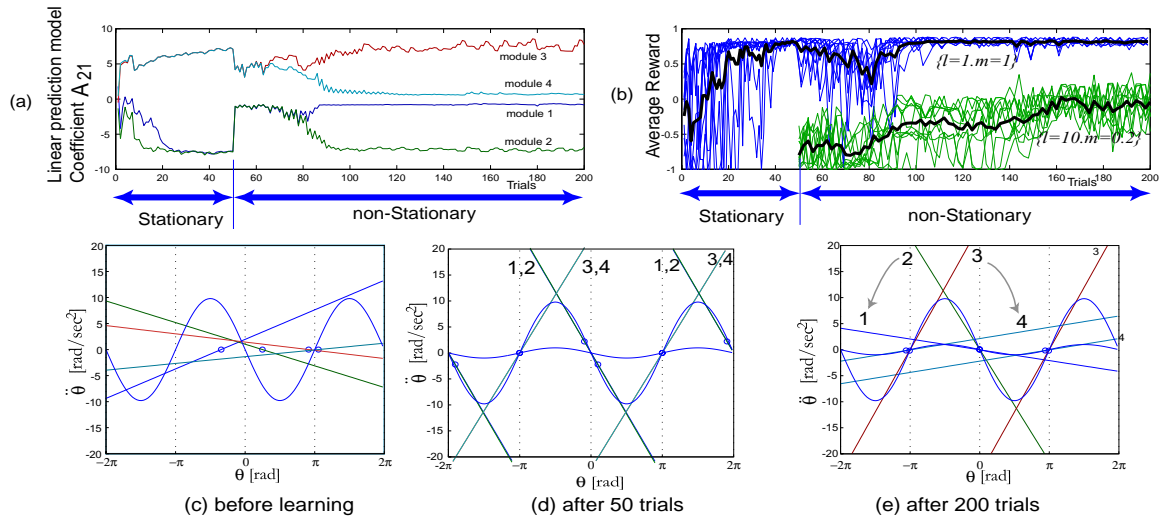


Figure 6: Time course of learning and changes of the prediction models. (a) Changes of a coefficient $A_{21} = \frac{\partial \ddot{\theta}}{\partial \theta}$ of the four prediction models. coefficient with angle. (b) Change of average reward during each trial. Thin lines: results of 10 simulation runs. Thick line: average to 10 simulation runs. Note that the average reward with the new, longer pendulum was lower even after successful learning because of its longer period of swinging. (c), (d), and (e): Linear prediction models in the section of $\{\theta = 0, T = 0\}$ before learning (c), after 50 trials with fixed parameters (d), and after 150 trials with changing parameters (e). Slopes of linear models correspond to A_{21} shown in (a).

implemented a stabilizing feedback controller.

In 50 to 200 trials, the parameters of the pendulum were switched between $\{m = 1, l = 1.0\}$ and $\{m = 0.2, l = 10.0\}$ in each trial. At first, the degenerated modules tried to follow the alternating environment (Figure 6(a)), and thus swing up was not successful for the new, longer pendulum. The performance for the shorter pendulum was also disturbed (Figure 6(b)). After about 80 trials, the prediction models gradually specialized in either new or learned dynamics (Figure 6(e)), and successful swing up was achieved both for the shorter and longer pendulums.

We found similar module specialization in six of ten simulation runs. In four other runs, due to the bias in initial module allocation, three modules were aggregated in one domain (top or bottom) and one model covered the other domain during the stationary condition. However, after 150 trials in the non-stationary condition, module specialization as shown in Figure 6(e) was achieved.

5 Discussion

We proposed a multiple model-based reinforcement learning (MMRL) architecture that decomposes a non-linear and/or non-stationary task in space and time based on the local predictability of the system dynamics. We tested the performance of the MMRL in both non-linear and non-stationary control tasks. It was shown in simulations of the pendulum swing up task that multiple prediction models as well as reinforcement learning controllers were successfully trained and specialized for different domains in the state space. It was also confirmed in a nonstationary pendulum swing-up task that available modules are flexibly allocated for different domains in space and time based on the task demands.

The modular control architecture using multiple prediction models was proposed by Wolpert and Kawato as a computational model of the cerebellum (Wolpert et al. 1998; Wolpert and Kawato 1998). Imamizu et al. showed in fMRI experiments of novel tool use that a large area of the cerebellum is activated initially and then a smaller area remains to be active after long training. They proposed that such local activation spots are the neural correlates of internal models of tools (Imamizu et al. 2000). They also showed that internal models of different tools are represented in separated areas in the cerebellum (Imamizu et al. 1997). Our simulation results in a non-stationary environment can provide a computational account of these fMRI data. When a new task is introduced, many modules initially compete to learn it. However, after repetitive learning, only a subset of modules are specialized and recruited for the new task.

One limitation of MLQC architecture is that the reward function has non-zero gradient

in the domains of each module. A method for back-propagating the value function of the successor module as the effective reward for the predecessor module is under development.

In order to construct a hierarchical RL system, it appears necessary to combine both top-down and bottom-up approaches for task decomposition. The MMRL architecture provides one solution for the bottom-up approach. Combination of this bottom-up mechanism with a top-down mechanism is the subject of our ongoing study.

Acknowledgments

We thank Masahiko Haruno, Daniel Wolpert, Chris Atkeson, Jun Tani, Hidenori Kimura, and Raju Bapi for helpful discussions.

References

- Barto, A. G., R. S. Sutton, and C. W. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13, 834–846.
- Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*. Belmont, MA, USA: Athena Scientific.
- Cacciatore, T. W. and S. J. Nowlan (1994). Mixture of controllers for jump linear and non-linear plants. In J. D. Cowan, G. Tesauero, and J. Alspector (Eds.), *Advances in Neural Information Processing System*, Volume 6. San Mateo, CA, USA: Morgan Kaufmann.
- Dayan, P. and G. E. Hinton (1993). Feudal reinforcement learning. In C. L. Giles, S. J. Hanson, and J. D. Cowan (Eds.), *Advances in Neural Information Processing Systems* 5, pp. 271–278. San Mateo, CA, USA: Morgan Kaufmann.
- Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation* 12, 243–269.
- Gomi, H. and M. Kawato (1993). Recognition of manipulated objects by motor learning with modular architecture networks. *Neural Networks* 6, 485–497.
- Haruno, M., D. M. Wolpert, and M. Kawato (1999). Multiple paired forward-inverse models for human motor learning and control. In M. S. Kearns, S. A. Solla, and D. A. Cohen (Eds.), *Advances in Neural Information Processing Systems*, 11, Cambridge, MA, USA, pp. 31–37. MIT Press.
- Imamizu, H., S. Miyauchi, Y. Sasaki, R. Takino, B. Pütz, and M. Kawato (1997). Separated modules for visuomotor control and learning in the cerebellum: A functional

- MRI study. In A. W. Toga, R. S. J. Frackowiak, and J. C. Mazziotta (Eds.), *NeuroImage: Third International Conference on Functional Mapping of the Human Brain*, Volume 5, Copenhagen, Denmark, pp. S598. Academic Press.
- Imamizu, H., S. Miyauchi, T. Tamada, Y. Sasaki, R. Takino, B. Putz, T. Yoshioka, and M. Kawato (2000). Human cerebellar activity reflecting an acquired internal model of a new tool. *Nature* 403, 192–195.
- Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). Adaptive mixtures of local experts. *Neural Computation* 3, 79–87.
- Littman, M., A. Cassandra, and L. Kaelbling (1995). Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russel (Eds.), *Machine Learning: Proceedings of the 12th International Conference*, San Francisco, CA, USA, pp. 362–370. Morgan Kaufmann.
- Narendra, K. S., J. Balakrishnan, and M. K. Ciliz (1995). Adaptation and learning using multiple models, switching and tuning. *IEEE Control Systems Magazine* June, 37–51.
- Parr, R. and S. Russel (1998). Reinforcement learning with hierarchies of machines. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems 10*, Cambridge, MA, USA, pp. 1043–49. MIT Press.
- Pawelzik, K., J. Kohlmorge, and K. R. Müller (1996). Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Computation* 8, 340–356.
- Schaal, S. and C. G. Atkeson (1996). From isolation to cooperation: An alternative view of a system of experts. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems 8*, pp. 605–611. Cambridge, MA, USA: MIT Press.
- Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8, 323–340.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal difference. *Machine Learning* 3, 9–44.
- Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning*. Cambridge, MA, USA: MIT Press.
- Sutton, R. S., S. Singh, D. Precup, and B. Ravindran (1999). Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems 11*. Cambridge, MA, USA: MIT Press.

- Tani, J. and S. Nolfi (1999). Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks 12*, 1131–1141.
- Wiering, M. and J. Schmidhuber (1998). HQ-learning. *Adaptive Behavior 6*, 219–246.
- Wolpert, D. M. and M. Kawato (1998). Multiple paired forward and inverse models for motor control. *Neural Networks 11*, 1317–1329.
- Wolpert, D. M., R. C. Miall, and M. Kawato (1998). Internal models in the cerebellum. *Trends in Cognitive Sciences 2*, 338–347.