Multiple Model-based Reinforcement Learning

Kenji Doya*[1235], Kazuyuki Samejima[1,3],

Ken-ichi Katagiri[45], and Mitsuo Kawato[135]

November 2, 2001

[1]Human Information Science Laboratories, ATR International

2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan

Phone: +81-774-95-1251

Fax: +81-774-95-1259

Email: doya@atr.co.jp

[2]CREST, Japan Science and Technology Corporation

[3]Kawato Dynamic Brain Project, ERATO

Japan Science and Technology Corporation

[4]ATR Human Information Processing Research Laboratories

[5]Nara Institute of Science and Technology

Abstract

We propose a modular reinforcement learning architecture for non-linear, non-stationary control tasks, which we call multiple model-based reinforcement learning (MMRL). 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. We formulate MMRL for both discrete-time, finite state case and continuous-time, continuous state case. The performance of MMRL was demonstrated for discrete case in a non-stationary hunting task in a grid world and for continuous case in a non-linear, non-stationary control task of swinging up a pendulum with variable physical parameters.

1 Introduction

A big issue in the application of reinforcement learning (RL) to real-world control problems is how to deal with non-linearity and non-stationarity. For a nonlinear, high-dimensional system, the conventional discretizing approach necessitates a huge number of states, which makes learning very slow. Standard RL algorithms can perform badly when the environment is non-stationary, or has hidden states. 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; Morimoto and Doya 2001). The basic problem in modular or hierarchical RL is how to decompose a complex task into simpler 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 originally 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). It was recently reformulated as "MOdular Selection and Identification for Control (MOSAIC)" (Wolpert and Ghahramani 2000).

In this paper, we apply the idea of a softmax selection of modules to the paradigm of reinforcement learning. The resulting learning architecture, which we call "Multiple Model-based Reinforcement Learning (MMRL)," learns to de-

compose a non-linear and/or non-stationary task through the competition and cooperation of multiple prediction models and reinforcement learning controllers.

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 first test the performance of the MMRL architecture for the discrete case in a hunting task with multiple preys in a grid world (Section 4). We also demonstrate the performance of MMRL for continuous case in a non-linear, non-stationary control task of swinging up a pendulum with variable physical parameters (Section 5).

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, finite state environment

$$P(x(t)|x(t-1),u(t-1)) = F(x(t),x(t-1),u(t-1)), \qquad (t=1,2,...), \quad (1)$$

where $x \in \{1,...,N\}$ and $u \in \{1,...,M\}$ are discrete states and actions, and a continuous-time, continuous-state environment

$$\dot{x}(t) = f(x(t),u(t)) + \nu(t), \qquad (t \in [0,\infty)), \quad (2)$$

where $x \in R^N$ and $u \in R^M$ are state and action vectors, and $\nu \in R^N$ is noise. Actions are given by a policy, either a stochastic one

$$P(u(t)|x(t)) = G(u(t),x(t)) \quad (3)$$

or a deterministic one

$$u(t) = g(x(t)). \quad (4)$$

The reward $r(t)$ is given as a function of the state $x(t)$ and the action $u(t)$. The goal of reinforcement learning is to improve the policy so that more rewards are acquired in a long run. The basic strategy of reinforcement learning is to estimate cumulative future reward under the current policy as the "value function" $V(x)$ for each state and then to improve the policy based on the value function. We define the value function of the state $x(t)$ under the current policy as

$$V(x(t)) = E\left[\sum_{k=0}^{\infty} \gamma^k r(t+k)\right] \quad (5)$$

in discrete case (Sutton and Barto 1998) and

$$V(x(t)) = E\left[\int_0^{\infty} e^{-\frac{s}{\tau}} r(t+s)ds\right] \quad (6)$$

in continuous case (Doya 2000), where $0 \leq \gamma \leq 1$ and $0 < \tau$ are the parameters for discounting future reward.

2.1 Responsibility Signal

The purpose of the prediction model in each module is to predict the next state (discrete-time) or the temporal derivative of the state (continuous-time) based on the observation of the state and the action. The responsibility signal $\lambda_i(t)$ (Wolpert and Kawato 1998; Haruno et al. 1999) is given by the relative goodness of predictions of multiple prediction models.

For a unified description, we denote the new state in the discrete case as

$$y(t) = x(t) \tag{7}$$

and the temporal derivative of the state in the continuous case as

$$y(t) = \dot{x}(t). \tag{8}$$

The basic formula for the responsibility signal is given by the Bayes rule

$$\lambda_i(t) = P(i|y(t)) = \frac{P(i)P(y(t)|i)}{\sum_{j=1}^{n} P(j)P(y(t)|j)}, \tag{9}$$

where $P(i)$ is the prior probability of selecting module i and $P(y(t)|i)$ is the likelihood of model i given the observation $y(t)$.

In the discrete case, the prediction model gives the probability distribution of the new state $\hat{x}(t)$ based on the previous state $x(t-1)$ and the action $u(t-1)$ as

$$P(\hat{x}(t)|x(t-1), u(t-1)) = F_i(\hat{x}(t), x(t-1), u(t-1)) \qquad (i = 1, ..., n). \tag{10}$$

If there is no prior knowledge about module selection, we take the priors as uniform ($P(i) = 1/n$) and then the responsibility signal is given by

$$\lambda_i(t) = \frac{F_i(x(t), x(t-1), u(t-1))}{\sum_{j=1}^{n} F_j(x(t), x(t-1), u(t-1))}, \tag{11}$$

where $x(t)$ is the newly observed state.

In the continuous case, the prediction model gives the temporal derivative of the state

$$\hat{\dot{x}}_i(t) = f_i(x(t), u(t)) \tag{12}$$

By assuming that the prediction error is Gaussian with variance σ^2, the responsibility signal is given by the Gaussian softmax function

$$\lambda_i(t) = \frac{e^{-\frac{1}{2\sigma^2}||\dot{x}(t)-\hat{\dot{x}}_i(t)||^2}}{\sum_{j=1}^{n} e^{-\frac{1}{2\sigma^2}||\dot{x}(t)-\hat{\dot{x}}_i(t)||^2}}, \tag{13}$$

where $\dot{x}(t)$ is the actually observed state change.

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 weighted by the responsibility signal $\lambda_i(t)$. In the discrete case, the prediction of the next state is given by

$$P(\hat{x}(t)) = \sum_{i=1}^{n} \lambda_i(t) F_i(\hat{x}(t), x(t-1), u(t-1)). \tag{14}$$

In the continuous case, the predicted state derivative is given by

$$\hat{\dot{x}}(t) = \sum_{i=1}^{n} \lambda_i(t) \hat{\dot{x}}_i(t). \tag{15}$$

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

2) Prediction model learning: The responsibility signal $\lambda_i(t)$ is also used for weighting the parameter update of the prediction models. In general, it is realized by scaling the error signal of prediction model learning by $\lambda_i(t)$.

3) Action output: The outputs of reinforcement learning controllers are linearly weighted by $\lambda_i(t)$ to make the action output. In the discrete case, the probability of taking an action $u(t)$ is given by

$$P(u(t)) = \sum_{i=1}^{n} \lambda_i(t) G_i(u(t), x(t)). \tag{16}$$

In the continuous case, the output is given by the interpolation of modular outputs

$$u(t) = \sum_{i=1}^{n} \lambda_i(t) u_i(t) = \sum_{i=1}^{n} \lambda_i(t) g_i(x(t)). \tag{17}$$

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,

$$\delta(t) = r(t) + \gamma V(x(t+1)) - V(x(t)) \tag{18}$$

in the discrete case and

$$\delta(t) = \hat{r}(t) - \frac{1}{\tau} V(t) + \dot{V}(t) \tag{19}$$

in the continuos case, is weighted by the responsibility signal

$$\delta_i(t) = \lambda_i(t) \delta(t) \tag{20}$$

for learning of 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

获得一个概率分布值

得到价值函数

its value function for the context under which its paired prediction model makes valid predictions.

2.3 Responsibility predictors

When there is some prior knowledge or belief about module selection, we incorporate the "responsibility predictors" (Wolpert and Kawato 1998; Haruno et al. 1999). By assuming their outputs $\hat{\lambda}_i(t)$ are proportional to the prior probability of module selection, from (9), the responsibility signal is given by

$$\lambda_i(t) = \frac{\hat{\lambda}_i(t) P(y(t)|i)}{\sum_{j=1}^{n} \hat{\lambda}_j(t) P(y(t)|j)}. \tag{21}$$

In modular decomposition of a task, it is desired that modules do not switch too frequently. This can be enforced by incorporating responsibility priors based on the assumption of temporal continuity and spatial locality of module activation.

2.3.1 Temporal continuity

The continuity of module selection is incorporated by taking the previous responsibility signal as the responsibility prediction signal. In the discrete case, we take the responsibility prediction based on the previous responsibility

$$\hat{\lambda}_i(t) = \lambda_i(t-1)^{\alpha}, \tag{22}$$

where $0 < \alpha < 1$ is a parameter that controls the strength of the memory effect. From (21) and (22), the responsibility signal at time t is given by the product of likelihoods of past module selection

$$\lambda_i(t) = \frac{1}{Z(t)} \prod_{k=0}^{t} P(x(t-k)|i)^{\alpha^k}, \tag{23}$$

where $Z(t)$ denotes the normalizing factor, i.e., $Z(t) = \sum_{j=1}^{n} \prod_{k=0}^{t} P(x(t-k)|j)^{\alpha^k}$.

In the continuous case, we choose the prior

$$\hat{\lambda}_i(t) = \lambda_i(t - \Delta t)^{\Delta t \alpha^{\Delta t}}, \tag{24}$$

where Δt is an arbitrarily small time difference (note (24) coincides with (22) with $\Delta t = 1$).

Since the likelihood of the module i is given by the Gaussian $P(\dot{x}(t)|i) = e^{-\frac{1}{2\sigma^2}||\dot{x}(t)-\hat{\dot{x}}_i(t)||^2}$, from recursion as in (23), the responsibility signal at time t is given by

$$\begin{aligned}\lambda_i(t) &= \frac{1}{Z(t)} \prod_{k=0}^{t/\Delta t} P(\dot{x}(t - k\Delta t)|i)^{\Delta t \alpha^{k\Delta t}} \\ &= \frac{1}{Z(t)} e^{-\frac{1}{2\sigma^2}\Delta t \sum_{k=0}^{t/\Delta t} ||\dot{x}(t-k\Delta t)-\hat{\dot{x}}_i(t-k\Delta t)||^2 \alpha^{k\Delta t}},\end{aligned} \tag{25}$$

that is, a Gaussian softmax function of temporally weighted squared errors. In the limit of $\Delta t \to 0$, (25) can be represented as

$$\lambda_i(t) = \frac{e^{-\frac{1}{2\sigma^2}E_i(t)}}{\sum_{j=1}^{n} e^{-\frac{1}{2\sigma^2}E_j(t)}} \tag{26}$$

where $E_i(t)$ is a low-pass filtered prediction error

$$\dot{E}_i(t) = \log \alpha \, E_i(t) + ||\dot{x}(t) - \hat{\dot{x}}_i(t)||^2. \tag{27}$$

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

2.3.2 Spatial locality

In the continuous case, we consider a Gaussian spatial prior

$$\hat{\lambda}_i(t) = \frac{e^{-\frac{1}{2}(x(t)-\mathbf{c}_i)' M_i^{-1} (x(t)-\mathbf{c}_i)}}{\sum_{j=1}^{n} e^{-\frac{1}{2}(x(t)-\mathbf{c}_j)' M_j^{-1} (x(t)-\mathbf{c}_j)}}, \tag{28}$$

where $\mathbf{c}_i$ is the center of the area of specialization, M_i is a covariance matrix that specifies the shape, and $'$ denotes transpose. These parameters are updated so that they approximate the distribution of the input state $x(t)$ weighted by the responsibility signal, namely,

$$\dot{\mathbf{c}}_i = \eta_c \lambda_i(t)(-\mathbf{c}_i + x(t)), \tag{29}$$

$$\dot{M}_i = \eta_M \lambda_i(t)[-M_i + (x(t) - \mathbf{c}_i)(x(t) - \mathbf{c}_i)'], \tag{30}$$

where η_c and η_M are update rates.

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, finite state and action problems. The standard way of utilizing a predictive model in RL is to use it for action selection by the one-step search

$$u(t) = \arg\max_u E[\hat{r}(x(t), u) + \gamma V(\hat{x}(t+1))], \tag{31}$$

where $\hat{r}(x(t), u)$ is the predicted immediate reward and $\hat{x}(t+1)$ is the next state predicted from the current state $x(t)$ and a candidate action u.

In order to implement this algorithm, we provide each module with a reward model $\hat{r}_i(x, u)$, a value function $V_i(x)$, and a dynamic model $F_i(\hat{x}, x, u)$. Each candidate action u is then evaluated by

$$\begin{aligned} q(x(t), u) &= E[\hat{r}(x(t), u) + \gamma V(\hat{x}(t+1))|u] \\ &= \sum_{i=1}^{n} \lambda_i(t)[\hat{r}_i(x(t), u) + \gamma \sum_{\hat{x}=1}^{N} V_i(\hat{x}) F_i(\hat{x}, x(t), u)]]. \end{aligned} \tag{32}$$

For the sake of exploration, we use a stochastic version of the greedy action selection (31), where the action $u(t)$ is selected by a Gibbs distribution

$$P(u|x(t)) = \frac{e^{\beta q(x(t),u)}}{\sum_{u'=1}^{M} e^{\beta q(x(t),u')}}, \tag{33}$$

where β controls the stochasticity of action selection.

The parameters are updated by the error signals weighted by the responsibility signal, namely, $\lambda_i(t)(F_i(j, x(t-1), u(t-1)) - c(j, x(t)))$ for the dynamic model ($j = 1, ..., N$; $c(j, x) = 1$ if $j = x$ and zero otherwise), $\lambda_i(t)(\hat{r}_i(x(t), u(t)) - r(t))$ for the reward model, and $\lambda_i(t)\delta(t)$ for the value function model.

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

$$\frac{1}{\tau} V(x(t)) = \max_u \left[r(x(t), u) + \frac{\partial V(x(t))}{\partial x} f(x(t), u) \right], \tag{34}$$

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

$$u = g\left(\frac{\partial f(x,u)}{\partial u}'\frac{\partial V(x)}{\partial x}'\right), \tag{35}$$

where $\frac{\partial V(x)}{\partial x}'$ is a vector representing the steepest ascent direction of the value function, $\frac{\partial f(x,u)}{\partial u}'$ 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 $f_i(x,u)$ and a value model $V_i(x)$. The outputs of the dynamic models (12) are compared with the actually observed state dynamics $\dot{x}(t)$ to calculate the responsibility signal $\lambda_i(t)$ according to (13).

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

$$\hat{\dot{x}}(t) = \sum_{i=1}^{n} \lambda_i(t) f_i(x(t), u(t)), \tag{36}$$

and value function estimation

$$V(x) = \sum_{i=1}^{n} \lambda_i(t) V_i(x). \tag{37}$$

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

$$u_i(t) = g\left(\frac{\partial f_i(x,u)}{\partial u}'\frac{\partial V_i(x)}{\partial x}'\right)\Bigg|_{x(t)}. \tag{38}$$

They are then weighted by $\lambda_i(t)$ according to (17) to make the actual action $u(t)$.

Learning is based on the weighted prediction errors $\lambda_i(t)(\hat{\dot{x}}_i(t) - \dot{x}(t))$ for dynamic models and $\lambda_i(t)\delta(t)$ for value function models.

3.3 Multiple Linear Quadratic Controllers

In a modular architecture like the MMRL, the use of universal non-linear function approximators with large numbers of 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

$$\hat{\dot{x}}(t) = A_i(x(t) - x_i^d) + B_i u(t) \tag{39}$$

and a local quadratic reward model

$$\hat{r}(x(t), u(t)) = r_i^0 - \frac{1}{2}(x(t) - x_i^r)' Q_i (x(t) - x_i^r) - \frac{1}{2} u'(t) R_i u(t) \tag{40}$$

for each module, where x_i^d, x_i^r is the center of local prediction for state and reward, respectively. The r_i^0 is a bias of quadratic reward model.

The value function is given by the quadratic form

$$V_i(x) = v_i^0 - \frac{1}{2}(x - x_i^v)' P_i (x - x_i^v) \tag{41}$$

The matrix P_i is given by solving the Riccati equation

$$0 = \frac{1}{\tau} P_i - P_i A_i - A_i' P_i + P_i B_i R_i^{-1} B_i' P_i - Q_i. \tag{42}$$

The center x_i^v and the bias v_i^0 of the value function is given by

$$x_i^v = (Q_i + P_i A_i)^{-1} (Q_i x_i^r + P_i A_i x_i^d), \tag{43}$$

$$\frac{1}{\tau} v_i^0 = r_i^0 - \frac{1}{2}(x_i^v - x_i^r)' Q_i (x_i^v - x_i^r) \tag{44}$$

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

$$u_i(t) = -R_i^{-1} B_i^T P_i (x(t) - x_i^v) \tag{45}$$

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

$$u(t) = \sum_{i=1}^{n} \lambda_i(t) u_i(t). \tag{46}$$

The parameters of the local linear models A_i, B_i, and x_i^d and those of the quadratic reward models r_i^0, Q_i, and R_i are updated by the weighted prediction errors $\lambda_i(t)(\hat{\dot{x}}_i(t) - \dot{x}(t))$ and $\lambda_i(t)(r_i(x,u) - r(t))$, respectively. When we assume that the update of these models is slow enough, then the Riccati equations (42) may be recalculated only intermittently. We call this method "Multiple Linear Quadratic Controllers" (MLQC).

4 Simulation: Discrete Case

In order to test the effectiveness of the MMRL architecture, we first applied the discrete MMRL architecture to a non-stationary hunting task in a grid world. The hunter agent tries to catch a prey in a 7×7 torus grid world. There are 47 states representing the position of the prey relative to the hunter. The hunter chooses one of five possible actions: { north (N), east (E), south (S), west (W), stay}. A prey moves in a fixed direction during a trial. At the beginning of each trial, one of four movement directions { NE, NW, SE, SW} is randomly selected and a prey is placed at a random position in the grid world. When the hunter catches the prey, by stepping into the same grid as the prey's, a reward $r(t) = 10$ is given. Each step of movement costs $r(t) = -1$. A trial is terminated when the hunter catches a prey, or it fails to catch it within 100 steps.

In order to compare the performance of MMRL with conventional methods, we applied standard Q-learning and compositional Q-learning (CQ-L) (Singh 1992) to the same task. A major difference between CQ-L and MMRL is the criterion for modular decomposition: CQ-L uses the consistency of the modular value functions while MMRL uses the prediction errors of dynamic models. In CQ-L, the gating network as well as component Q-learning modules are trained so that the composite Q-value well approximates the action value function of the entire problem. In the original CQ-L (Singh 1992), the output of the gating network was based on the "augmenting bit" that explicitly signaled the change in the context. Since our goal now is to let the agent learn appropriate decomposition of the task without an explicit cue, we used a modified CQ-L (See Appendix for the details of the algorithm and the parameters).

4.1 Result

Figure 2 shows the performance difference of standard Q-learning, CQ-L, and MMRL in the hunting task. The modified CQ-L did not perform significantly better than standard, flat Q-learning. Investigation of the modular Q functions of CQ-L revealed that, in most simulation runs, modules did not appropriately differentiate for four different kinds of preys. On the other hand, the performance of MMRL approached close to theoretical optimum. This was because four modules successfully specialized in one of four kinds of prey movement.

Figure 3 shows examples of the value functions and the prediction models learned by MMRL. From the output of the prediction models F_i, it can be seen that the modules 1,2,3 and 4 were specialized for the prey moving to NE, NW, SW, and SE, respectively. The landscapes of the value functions $V_i(x)$ are in

accordance with these movement directions of the prey.

A possible reason for the difference in the performance of CQ-L and MMRL in this task is the difficulty of module selection. In CQ-L, when the prey is far from the hunter, the differences in discounted Q values for different kinds of preys are minor. Thus it would be difficult to differentiate modules based solely on the Q values. In MMRL, on the other hand, module selection based on the state change, in this case prey movement, is relatively easy even when the prey is far from the hunter.

5 Simulation: Continuous Case

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 4) (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. $x = (\theta, \dot{\theta})' \in [-\pi, \pi] \times \mathbf{R}$ where θ is the joint angle ($\theta = 0$ means the pendulum hanging down). The action was $u = T$. The reward was given by the height of the tip and the negative squared torque

$$r(x, u) = -\cos\theta - \frac{1}{2}RT^2. \tag{47}$$

A trial was started from random joint angle $\theta \in [-\pi/4, \pi/4]$ with no angular velocity, and lasted for 20 seconds. We devised the following automatic annealing process for the parameter σ of the softmax function for the responsibility signal (26).

$$\sigma_{k+1} = \eta a E_k + (1 - \eta)\sigma_k, \tag{48}$$

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 $\sigma_0 = 4$.

5.1 Task Decomposition in Space: Non-linear Control

We first used two modules, each of which had a linear dynamic model (39) and a quadratic reward model (40). 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 a random position of the pendulum and lasted for 30 seconds.

Figure 4 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 5 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 first model predicted the negative feedback acceleration around the equilibrium state with the pendulum hanging down. The second model predicted the positive feedback acceleration around the unstable equilibrium with the pendulum raised up. The two reward models also approximated the cosine reward function using parabolic curves.

Figure 6 shows the dynamic and reward models when there were eight mod-

ules. 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 7 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. The addition of redundant modules resulted in more variability in the time course of learning. This is because there were multiple possible ways of modular decomposition and due to the variability of the sample trajectories, it took longer time for modular decomposition to stabilize. Nevertheless, learning by the 8-module MLQC was still much faster than by 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 (42). The controller at the bottom is a positive feedback controller that de-stabilizes the equilibrium where the reward is minimal, 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.

5.2 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 (39) and a quadratic reward model (40). The centers 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 8(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 8(b). Figures 8(c,d,e) show prediction models in the section of $\{\theta = 0, T = 0\}$.

Initial position gains are set randomly (Figure 8(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 8(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 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 8(a)), and thus swing up was not successful for the new, longer pendulum. The performance for the shorter pendulum was also disturbed (Figure 8(b)). After about 80 trials, the prediction models gradually specialized in either new or learned dynamics (Fig-

ure 8(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 8(e) was achieved.

6 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 were successfully trained and corresponding model-based controllers were derived.

The modules were 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 suggested 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 might argue whether MLQC is a reinforcement learning architecture since it uses LQ controllers that were calculated off-line. However, when the linear dynamic models and quadratic reward models are learned on-line, as in our simulations, the entire system realize reinforcement learning. One limitation of MLQC architecture is that the reward function should have helpful gradients in each modular domain. 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.

Appendix: Modified Compositional Q-learning

On each time step, the gating variable $g_i(t)$ is given by the prior probability of module selection, in this case from the assumption of temporal continuity (22)

$$g_i(t) = \frac{\lambda_i(t-1)^\alpha}{\sum_{j=1}^n \lambda_j(t-1)^\alpha}. \tag{49}$$

The composite Q-values for state $x(t)$ are then computed by

$$\hat{Q}(x(t), u) = \sum_{i=1}^n g_i(t) Q_i(x(t), u) \tag{50}$$

and an action $u(t)$ is selected by

$$P(u|x(t)) = \frac{e^\beta \hat{Q}(x(t), u)}{\sum_{v \in U} e^\beta \hat{Q}(x(t), v)}. \tag{51}$$

After the reward $r(x(t), u(t))$ is acquired and the state changes to $x(t+1)$, the TD error for the module i is given by

$$e_i(t) = r(x(t), u(t)) + \gamma \max_u \hat{Q}(x(t+1), u) - Q_i(x(t), u(t)). \tag{52}$$

From Gaussian assumption of value prediction error, the likelihood of module i is given by

$$P(e_i(t)|i) = e^{-\frac{1}{2\sigma^2} e_i(t)^2} \tag{53}$$

and thus the responsibility signal, or the posterior probability for selecting module i, is given by

$$\lambda_i(t) = \frac{g_i(t) e^{-\frac{1}{2\sigma^2} e_i(t)^2}}{\sum_j g_j(t) e^{-\frac{1}{2\sigma^2} e_j(t)^2}}. \tag{54}$$

The Q values of each module is updated with the weighted TD error $\lambda_i(t) e_i(t)$ as the error signal.

The discount factor was set as $\gamma = 0.9$ and the greediness parameters as $\beta = 1$ for both MMRL and CQ-L. The decay parameter of temporal responsibility predictor was $\alpha = 0.8$ for MMRL. We tried different values of α for CQ-L without a success. The value used in Figures 2 and 3 was $\alpha = 0.99$.

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. Tesauro, 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*, 215–245.

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.

Morimoto, J. and K. Doya (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems 36*, 37–51.

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., D. Precup, and S. Singh (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112*, 181–211.

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.

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 Z. Ghahramani (2000). Computational principles of movement neuroscience. *Nature Neuroscience 3*, 1212–1217.

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.

Figure Legends

Figure 1

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

Figure 2

Comparison of the performance of standard Q-learning (gray line), modified CQ-L (dashed line), and MMRL (thick line) in the hunting task. The average number of steps needed for catching a prey during 200 trial epochs in 10 simulation runs are plotted. The dash-dotted line shows the theoretically minimal average steps required for catching the prey.

Figure 3

Example of value functions and prediction models learned by MMRL after 10000 trials. Each slot in the grid shows the position of the prey relative to the hunter, which was used as the state x. (a) The state value functions $V_i(x)$. (b) The prediction model outputs $F_i(\hat{x}, x, u)$, where current state x of the prey was fixed as $(2, 1)$, shown by the circle, and the action u was fixed as "stay."

Figure 4

(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[rad],0.1[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 5

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 6

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

Figure 7

Learning curves for the pendulum swing up task. The cumulative reward $\int_0^{20} r(t)dt$ during each trial is shown for five simulation runs. (a) n=2 modules. (b) 4 modules. (c) 8 modules. (d) non-modular architecture.

Figure 8

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).

Figures

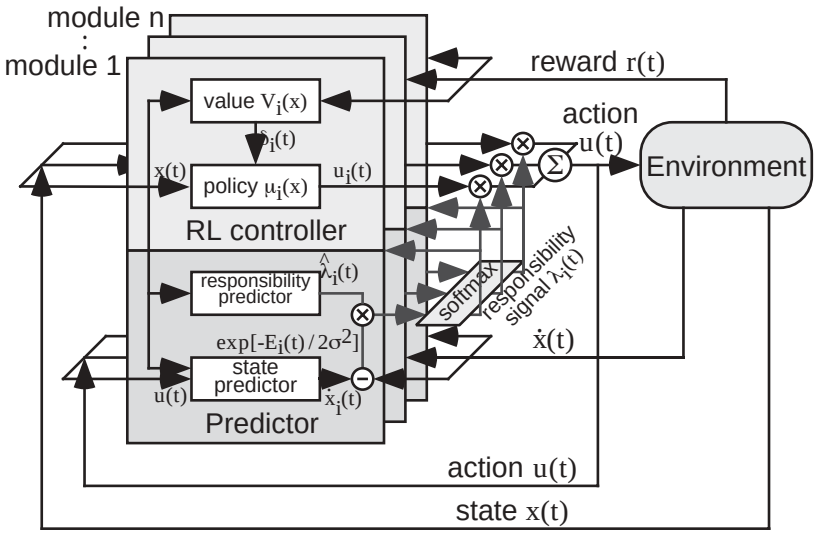

Figure 1: Kenji Doya, 2265.

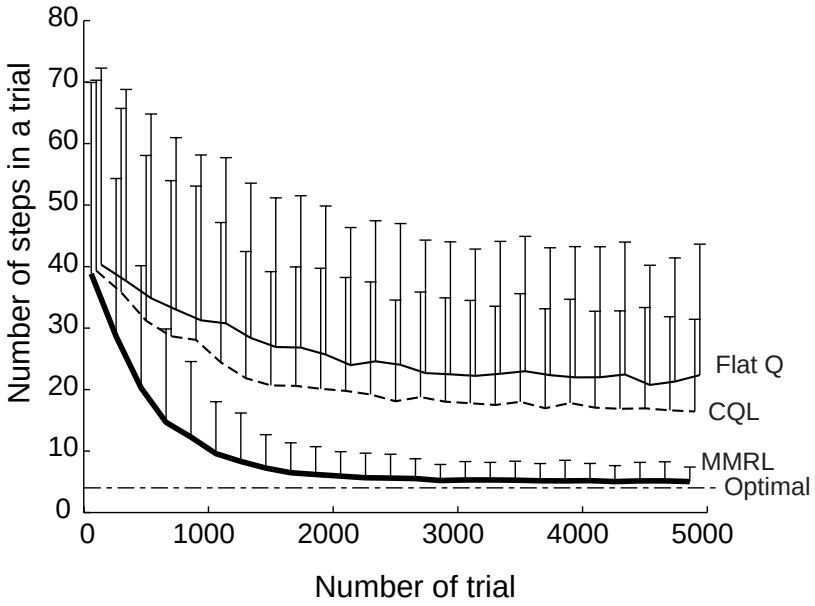

Figure 2: Kenji Doya, 2265.

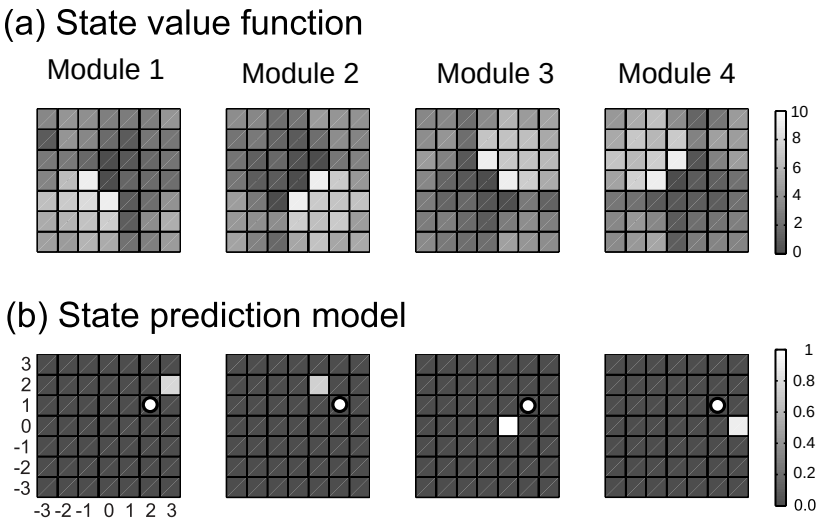

Figure 3: Kenji Doya, 2265.

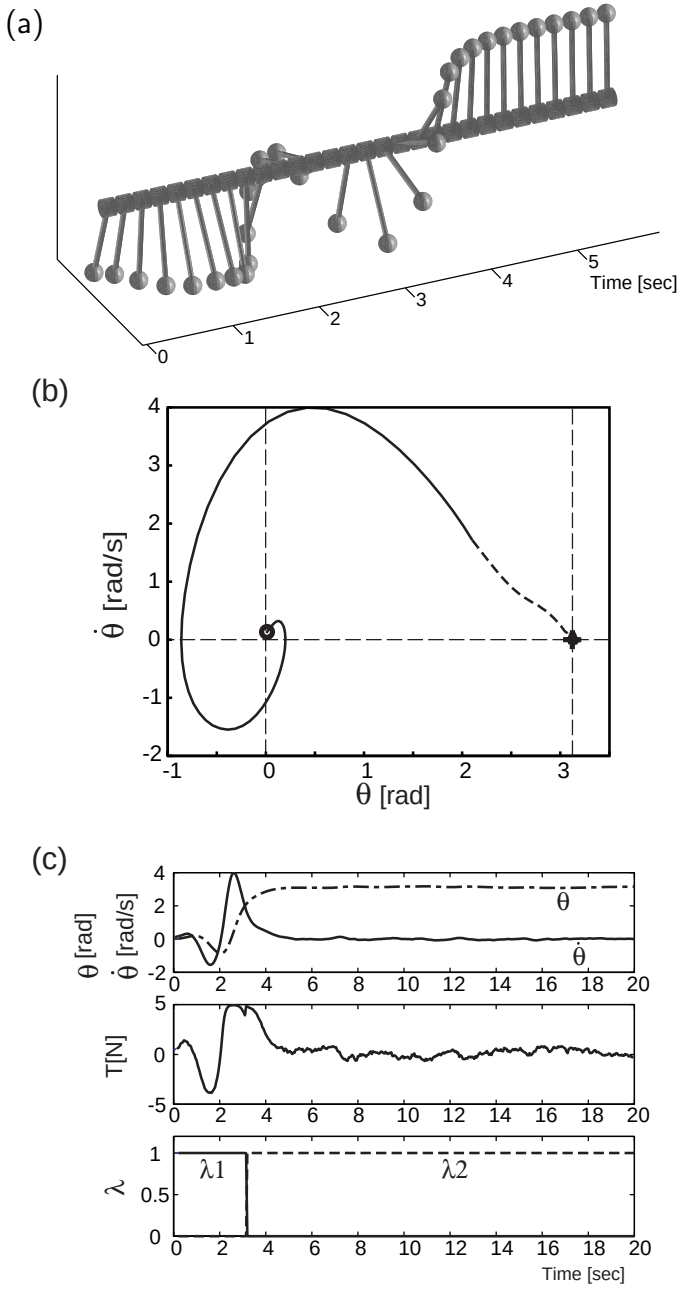

Figure 4: Kenji Doya, 2265.

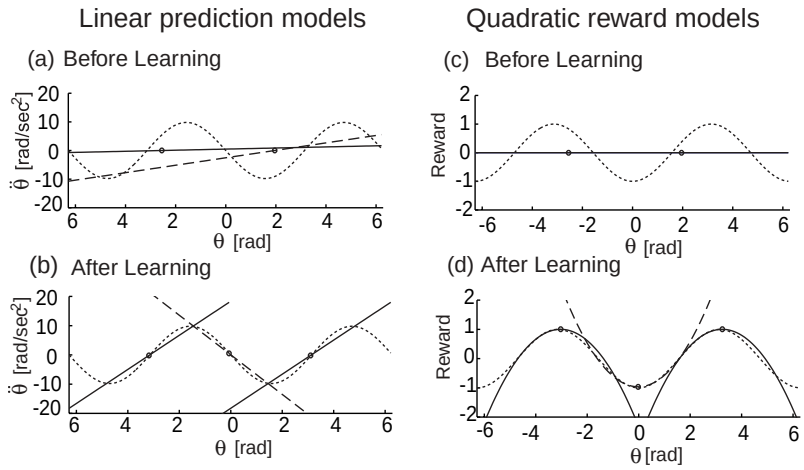

Figure 5: Kenji Doya, 2265.

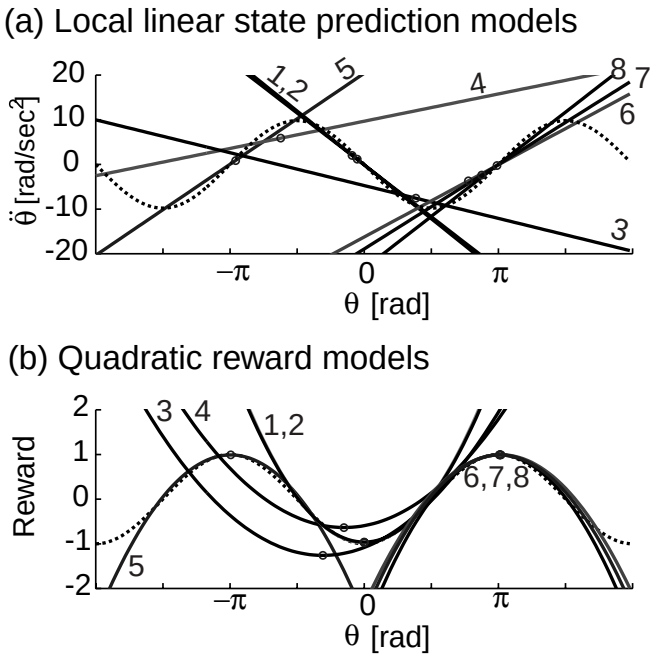

Figure 6: Kenji Doya, 2265.

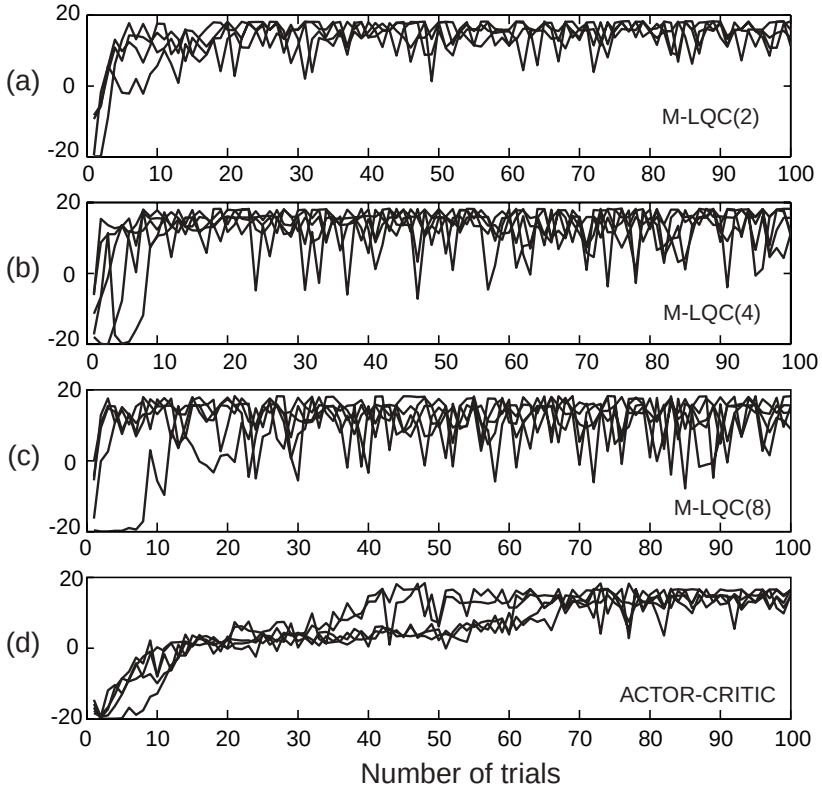

Figure 7: Kenji Doya, 2265.

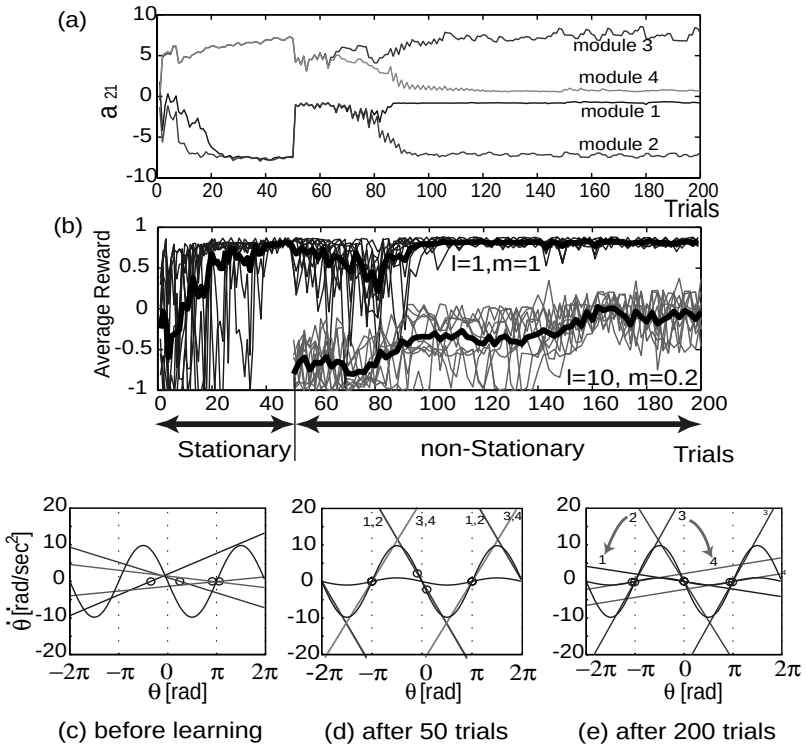

Figure 8: Kenji Doya, 2265.