Neural Networks 129 (2020) 149–162

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks

Dongqi Han[a], Kenji Doya[b], Jun Tani[a,*]

[a] Cognitive Neurorobotics Research Unit, Okinawa Institute of Science and Technology, Okinawa, Japan
[b] Neural Computation Unit, Okinawa Institute of Science and Technology, Okinawa, Japan

ARTICLE INFO

Article history:
Received 26 November 2019
Received in revised form 25 May 2020
Accepted 2 June 2020
Available online 6 June 2020

Keywords:
Recurrent neural network
Reinforcement learning
Partially observable Markov decision process
Multiple timescale
Compositionality

ABSTRACT

Recurrent neural networks (RNNs) for reinforcement learning (RL) have shown distinct advantages, e.g., solving memory-dependent tasks and meta-learning. However, little effort has been spent on improving RNN architectures and on understanding the underlying neural mechanisms for performance gain. In this paper, we propose a novel, multiple-timescale, stochastic RNN for RL. Empirical results show that the network can autonomously learn to abstract sub-goals and can self-develop an action hierarchy using internal dynamics in a challenging continuous control task. Furthermore, we show that the self-developed compositionality of the network enhances faster re-learning when adapting to a new task that is a re-composition of previously learned sub-goals, than when starting from scratch. We also found that improved performance can be achieved when neural activities are subject to stochastic rather than deterministic dynamics.

1. Introduction

Reinforcement learning (RL) (Sutton & Barto, 1998) with neural networks as function approximators, i.e., deep RL, has undergone rapid development in recent years. State-of-the-art RL frameworks have shown proficient performance in various kinds of tasks, from game playing (Mnih et al., 2016; Silver et al., 2016, 2017) to continuous robot control (Haarnoja, Zhou, Abbeel, & Levine, 2018; Lillicrap et al., 2015; Wang et al., 2017). While most deep RL studies have employed feed-forward neural networks (FNNs) to solve tasks that can be well modeled by Markov Decision Processes (MDP) (Bellman, 1957), RL with recurrent neural networks (RNNs) has garnered increasing attention (Al-Shedivat et al., 2018; Ha & Schmidhuber, 2018; Hausknecht & Stone, 2015; Heess, Hunt, Lillicrap, & Silver, 2015; Jaderberg et al., 2019; Kapturowski, Ostrovski, Dabney, Quan, & Munos, 2018; Shibata & Sakashita, 2015; Vezhnevets et al., 2017; Wang et al., 2018; Zhang, McCarthy, Finn, Levine, & Abbeel, 2016).

One benefit of RNNs comes from their capacity to solve a kind of *Partially Observable MDPs* (POMDP) (Åström, 1965) in which optimal decision-making requires information derived from historical observations, i.e. memory-dependent tasks. While the curse of dimensionality (Friedman, 1997) occurs if these tasks are modeled into MDPs by including all historic information in the current state, a more tractable way of solving memory-dependent tasks is to leverage the contextual capacity of RNNs as function approximators (Schmidhuber, 1991). Heess et al. (2015), Utsunomiya and Shibata (2008) and Wierstra, Foerster, Peters, and Schmidhuber (2007) have shown how RNNs enable successful RL in memory-dependent control tasks. Interestingly, even for tasks that are readily solved by MDPs, such as Atari games (Mnih et al., 2015), extraordinary performance can be achieved using relatively simple algorithms with RNNs (Kapturowski et al., 2018).

Furthermore, RNNs advance *meta-learning* in RL, defined as an effect that an agent requires statistically less time in learning to solve new tasks, compared to previously learned tasks, provided that the two tasks share some common elements (Frans, Ho, Chen, Abbeel, & Schulman, 2018; Thrun & Pratt, 1998; Wang et al., 2018). Al-Shedivat et al. (2018) showed meta-learning by robotic agents in dynamically changing tasks using recurrent policies, while (Wang et al., 2018) argued that the prefrontal cortex, which contains many recurrent connections, plays an important role in meta-learning.

Despite the success of RL with RNN in relatively simple environments, solving more sophisticated tasks often requires cognitive competency in dealing with hierarchical operations, such as for composition/decomposition of an entire task from a sequence of sub-goals (Bacon, Harb, & Precup, 2017; Dietterich, 2000; Sutton, Precup, & Singh, 1999). But very few studies have been devoted to developing hierarchical control utilizing RNN architectures. Vezhnevets et al. (2017) showed that multiple levels of RNN controllers with different temporal resolutions can

* Correspondence to: Cognitive Neurorobotics Research Unit, A626 (Level A, Lab 2), Okinawa Institute of Science and Technology Graduate University, 1919-1 Tancha, Onna-son, Okinawa, 904-0495, Japan.
E-mail address: jun.tani@oist.jp (J. Tani).

https://doi.org/10.1016/j.neunet.2020.06.002

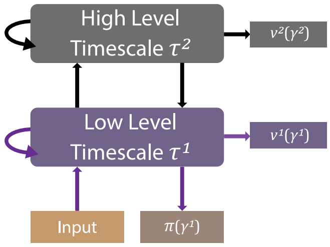

Fig. 1. The basic structure of MTSRNN is shown for the case of a 2-level configuration, used in this work. However, additional levels can readily be stacked onto it.

achieve dramatic performance on difficult hierarchical RL tasks. However, their method requires one to assume a particular form of transition model for embedding sub-goals. By contrast, our brains are good at self-developing action hierarchies for various tasks and also take advantage of them, which raises a question in our mind: are there any more basic neural mechanisms for discovering an action hierarchy?

Considering this, Yamashita and Tani (2008) introduced a multiple timescale RNN (MTRNN) containing fast-context and slow-context neurons, which was inspired by the ideas from cognitive science that the multiple timescales are essential to solve complicated cognitive tasks (Huys, Daffertshofer, & Beek, 2004; Newell, Liu, & Mayer-Kress, 2001; Smith, Ghazizadeh, & Shadmehr, 2006). They conducted an experiment in which a humanoid robot learned to generate different motor behavior to operate an object, from supervised samples. Although the explicit hierarchical structure of the task were not given, it was shown that the slow-context neurons autonomously learned to represent abstracted action primitives, such as touching and shaking the object. However, animals usually do not only extract hierarchies from supervised samples, but also can develop functional action primitives through trial-and-error (Badre & Frank, 2011; Badre, Kayser, & D'Esposito, 2010), which should be modeled by exploration-based RL. Moreover, we wondered how the learned action primitives can enhance learning novel tasks composed of previously learned sub-goals, which was not discussed in Yamashita and Tani (2008).

To this end, the current paper proposes a novel multi-timescale RNN architecture and an off-policy actor–critic algorithm for learning with multiple discount factors. We refer to our framework as *Recurrent Multi-timescale Actor–critic with STochastic Experience Replay* (**ReMASTER**). We also designed a sequential compositional task for testing the performance of the framework. Two essential proposals in this framework are as follows.

The first is to employ a multiple timescale property in neural activation dynamics (Chaudhuri, Knoblauch, Gariel, Kennedy, & Wang, 2015; Huys et al., 2004; Murray et al., 2014; Newell et al., 2001; Runyan, Piasini, Panzeri, & Harvey, 2017; Smith et al., 2006), as well as in the discount factors across different levels in an RNN. Although it has been shown that introduction of multiple-timescale neural activation dynamics in RNNs enhances development of hierarchy in supervised learning (Yamashita & Tani, 2008), such a possibility in RL remains to be investigated. In most RL algorithms, the discount factor (for an MDP) is treated as a single hyper-parameter. However, Enomoto et al. (2011) and Tanaka et al. (2016) have shown that dopamine neurons in mammalian brains encode value functions with different region-specific discount factors. In considering motor control, it is intuitive that detailed motor skills are learned with a faster discounting (on the order of seconds), while abstracted actions for long-term plans require longer timescales. In summary, it is expected that more detailed information processing can autonomously develop at lower levels by incorporating the faster timescale constraints imposed on both neural activation dynamics and the reward discounting. Meanwhile, more abstracted action plans can develop at higher levels with slower timescale constraints.

The second proposal is to introduce stochasticity not only in motor outputs, but also in internal neural dynamics at all levels of RNN for generating exploratory behaviors. This is inspired by the fact that cortical neurons, which play a key role in intelligence, have highly stochastic firing behaviors, both for irregular inter-spike intervals and for noisy firing rates (Beck et al., 2008; Beck, Ma, Pitkow, Latham, & Pouget, 2012; Hartmann, Lazar, Nessler, & Triesch, 2015; Softky & Koch, 1993). Chung et al. (2015) and Fraccaro, Sønderby, Paquet, and Winther (2016) have shown that various types of stochastic RNN models can learn to extract probabilistic structure hidden in temporal patterns by using variational Bayes approaches in supervised learning. It was also shown that stochastic FNNs facilitate efficient exploration and improved performance in RL (Florensa, Duan, & Abbeel, 2017; Fortunato et al., 2018). Therefore, we are interested in whether and how stochastic RNNs promote exploration and extraction of task features.

ReMASTER integrates these two essential ideas (multiple-timescale property and neuronal stochasticity) with an off-policy advantage actor–critic algorithm, in a model-free manner. We considered a kind of sequential, compositional tasks in which an agent learns to accomplish a set of sub-goals in a specific sequence without being given prior knowledge about the sub-goals. The experimental results using ReMASTER showed that compositionality develops autonomously, accompanied by an emergence of hierarchical representation of actions in the network. More specifically, action primitives for achieving task-relevant sub-goals were acquired in the lower level, characterized by faster timescale dynamics, whereas representation of those sub-goals was observed at the higher level characterized by the slower one. As a consequence of such self-developed hierarchical action control, we can "manipulate" the agent to consistently pursuing an undesired sub-goal by clamping high-level RNN states, analogous to animal optogenetic experiments (Morandell & Huber, 2017; Ramirez et al., 2013).

We further examined performance of ReMASTER by considering a multi-phase relearning task wherein an agent is required to adapt consecutively to new tasks that constitute a re-composition of previous-occurred sub-goals. ReMASTER outperformed other alternatives by showing remarkable performance in relearning cases because it was able to take advantage of previously learned representation about the sub-goals in a compositional way, thanks both to multiple timescales and neuronal stochasticity used in the model.

2. Methods

We used a multi-level stochastic RNN with level-specific timescales, as a basic network architecture for implementing ReMASTER. This architecture is referred to as *Multiple Timescale Stochastic Recurrent Neural Network* (MTSRNN). Fig. 1 shows the case of a 2-level MTSRNN where and γ^l represents the characteristic discount factor at lth level. v^l as the value function at lth level is estimated by temporal difference learning (TD-learning) (Sutton & Barto, 1998) using the corresponding γ^l. The policy function with discount factor γ^1, indicated by π, is estimated by the lowest level. Also, only the lowest level receives inputs. Note that although the network has multiple timescales of discounting, policy is improved to maximize expected return w.r.t. the lowest discounter factor γ^1. Learning the higher-level value function(s) $v^{l>1}$ serves as an auxiliary objective, which, nonetheless, we found critical in our experiments.

2.1. Multiple timescale stochastic RNN

Here we describe detailed mechanisms of L-levels MTSRNN. We use a super-script $l \in \{1, 2, \ldots, L\}$ to indicate the lth level, where a smaller l indicates a lower level. Let $\boldsymbol{u}$ and $\boldsymbol{c}$ denote the *hidden states* and the *RNN outputs*, respectively,[1] we have

$$\boldsymbol{u}^l(t) = (1 - \frac{1}{\tau^l})\boldsymbol{u}^l(t-1) + \frac{1}{\tau^l}\left[W_{cu}^{l-1,l}\boldsymbol{c}^{l-1}(t) + W_{cu}^{l,l}\boldsymbol{c}^l(t-1) + W_{cu}^{l+1,l}\boldsymbol{c}^{l+1}(t-1) + \boldsymbol{b}_u^l\right], \quad (1)$$

$$\boldsymbol{c}^l(t) = \tanh\left(\boldsymbol{u}^l(t) + \epsilon^l \boldsymbol{\sigma}^l(t)\right), \quad (2)$$

where $\boldsymbol{c}^{l-1}(t) = \boldsymbol{s}(t)$ when $l = 1$ is the current sensory input (state) and $\boldsymbol{c}^{l+1}$ does not exist for $l = L$. The scale of neuronal noise, σ^l, can be either a hyper-parameter or adaptive, and $\boldsymbol{\epsilon}^l(t)$ is a diagonal-covariance unit-Gaussian noise, which leads to a stochastic variable $\boldsymbol{c}^l(t)$ using the reparameterization trick (Kingma & Welling, 2013). The hyper-parameter τ^l is known as *timescale* of the lth level, which determines how fast hidden states vary, for which we usually have $\tau^l < \tau^{l+1}$. Synaptic weights and biases, denoted by W and $\boldsymbol{b}$, respectively, are trainable parameters of the neural network.

Value functions can be estimated via a linear connection from each level of the MTSRNN: $v^l(t) = (\boldsymbol{w}_{cv}^l)^T \boldsymbol{c}^l(t) + b_{cv}^l$. We focus on continuous action space, so the policy function can be expressed as diagonal Gaussian distributions $\boldsymbol{\pi}(t) \sim \mathcal{N}(\boldsymbol{p}(t), \boldsymbol{e}(t))$, where $\boldsymbol{p}(t) = \tanh\left(W_{ca}\boldsymbol{c}^1(t) + \boldsymbol{b}_{ca}\right)$ is the expected action assuming that the range of possible actions is $[-1, 1]$, and that $\boldsymbol{e}(t)$ is the exploration noise scale.

2.2. Off-policy advantage actor–critic

Recent deep RL studies using actor–critic algorithms with experience replay have achieved remarkable performance in many RL environments (Haarnoja et al., 2018; Lillicrap et al., 2015; Wang et al., 2017) by learning repetitively from previous experience. Therefore, for better sample efficiency, we use an off-policy actor–critic algorithm (Degris, White, & Sutton, 2012), which can deal with both continuous and discrete action spaces, although we consider continuous control in this work.

Suppose that in each episode, the agent is continuously interacting with the environment. At every step t, it experiences a state transition, which can be described by a tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t, done_t, \pi_t)$, where $\boldsymbol{s}, \boldsymbol{a}, r, \pi$ are state (observation), action, reward and policy function, respectively; and the Boolean $done_t$ indicates whether the episode ends at step $t + 1$. The agent stores the state transition in a replay buffer. In practice, RNNs require initial states for computing succeeding time development of RNN states. We set initial RNN states to zero at the beginning of each episode. For easier access to initial states during experience replay using RNN, we also recorded $\boldsymbol{c}^l(t)$ and $\boldsymbol{u}^l(t)$ of the MTSRNN at each step and used them in experience replay.[2]

For estimation of the critic, we used an off-policy version of the temporal difference (TD) learning algorithm to train value functions of all levels (Degris et al., 2012; Sutton & Barto, 1998). Knowing that (i) each level has a characteristic timescale τ^l; (ii) $1/(1-\gamma)$ indicates the eigen-timescale of discounting (Doya, 2000), it is natural to set the values of discount factors as

Algorithm 1 ReMASTER

```
Initialize the MTSRNN $\mathcal{R}$ and the replay buffer $\mathcal{B}$, global step $t \leftarrow 0$
repeat
    Reset an episode, assign $\mathcal{R}$ with zero initial RNN states
    while episode not terminated do
        Compute 1-step forward of $\mathcal{R}$ to obtain $(\boldsymbol{u}_t^l, \boldsymbol{c}_t^l)$
        Sample an action $\boldsymbol{a}_t$ from policy $\pi_t(\boldsymbol{a}|\boldsymbol{c}_t^1)$ and execute $\boldsymbol{a}_t$
        Obtain $\boldsymbol{s}_{t+1}$, $r_t$ and $done_t$ from the environment
        Record $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t, done_t)$, $\pi_t = \pi(\boldsymbol{a}_t|\boldsymbol{c}_t^1)$ and $(\boldsymbol{u}_t^l, \boldsymbol{c}_t^l)$ into $\mathcal{B}$
        if $mod(t, train\_interval) == 0$ then
            Sample sequential training samples to update $\mathcal{R}$ by Eqs. (4) and (7)
        end if
        $t \leftarrow t + 1$
    end while
until training stopped
```

$$\gamma^l = 1 - \frac{K}{\tau^l}, \quad (3)$$

where K is a constant to which we assigned a value of 0.16 throughout this work.

Let $\boldsymbol{\theta}$ denote the synaptic weights of the network. At each update, we randomly sample N state transition tuples from memory, and then conduct gradient descent for value functions v^l with learning rate α_v,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_v \frac{1}{L}\frac{1}{N}\sum_l^L \sum_i^N \left[\rho_i \delta_i^l \nabla_{\boldsymbol{\theta}} v^l(\boldsymbol{s}_{0:t_i}; \boldsymbol{\theta})\right], \quad (4)$$

where we have

$$\rho_i = \frac{\pi(\boldsymbol{a}_{t_i}|\boldsymbol{s}_{0:t_i}; \boldsymbol{\theta})}{\pi_{t_i}} \quad (5)$$

indicating the importance sampling ratio of the ith sample, where π_{t_i} is the behavior policy obtained from the replay buffer; and

$$\delta_i^l = r_{t_i} + \gamma^l v^l(\boldsymbol{s}_{0:t_i+1}; \boldsymbol{\theta}) - v^l(\boldsymbol{s}_{0:t_i}; \boldsymbol{\theta}) \quad (6)$$

is the TD-error for the ith sample and the lth level. Note that the value function v^l and the policy function π depend on $\boldsymbol{s}_{0:t_i}$ so that backpropagation through time (BPTT) is performed to calculate the gradients. They can also be written as $v^l(\boldsymbol{c}_{t_i}^l; \boldsymbol{\theta})$ and $\pi(\boldsymbol{a}|\boldsymbol{c}_{t_i}^1; \boldsymbol{\theta})$, respectively, if $\boldsymbol{c}_{t_i}^l$ has been computed.

To update the policy function, an advantage policy gradient algorithm was used in an off-policy manner (Degris et al., 2012), where the advantage was estimated by 1-step TD error with discount factor γ^1.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \frac{1}{N}\sum_i^N \left[\rho_i \delta_i^1 \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{a}_{t_i}|\boldsymbol{s}_{0:t_i}; \boldsymbol{\theta})\right], \quad (7)$$

where α_a is the learning rate for the actor. Algorithm 1 shows the overall procedure of ReMASTER. We followed algorithm 1 for all experiments in this work.

3. Results

We applied 2-levels ReMASTER to a so-called *sequential target-reaching task*. The following explains the details of task designs and simulation results and their analysis for a basic task, followed by those for an extended task that deals with relearning of consecutive task wherein goal changes.

[1] We collectively refer to $(\boldsymbol{u}, \boldsymbol{c})$ as *RNN states*.

[2] The recorded RNN states can be incompatible with the current RNN as learning goes on. However, this issue does not impact learning performance much because very old samples are discarded due to limited memory. We took this approach because of its simplicity (More discussion in Appendix A.3).

Table 1
Hyper-parameters we used in the sequential goal reaching task and the consecutive relearning task for ReMASTER. The Hyper-parameters with given search range were obtain by random search, otherwise hand-tuned.

Hyper-parameter	Description	Value	Search Range
γ^1	Low-level discount factor	0.92	
γ^2	High-level discount factor	0.98	
τ^1	Low-level RNN timescale	2	
τ^2	High-level RNN timescale	8	
N^1	Number of neurons in the lower level	100	[50, 200]
N^2	Number of neurons in the higher level	50	[25, 100]
buffer_size	Number of steps recorded in memory	5e6	
σ_0	Initial scale of neuronal noise	0.2	Appendix A.4
n_update	Number of steps per update	2	
lr_critic	Learning rate of critic	3e-4	[5e-5, 6e-4]
lr_actor	Learning rate of actor	1e-4	[1.5e-5, 2e-4]
α	Decay of the RMSProp optimizers	0.99	
batch_size	Number of training sequences.	16	
L	Sequence length for truncated BPTT	25	[10, 40]

3.1. Implementation details

We used an MTSRNN with 2 levels for ReMASTER in all experiments, where $\tau^1 = 2$ and $\tau^2 = 8$. The discount factors are $\gamma^1 = 0.92, \gamma^2 = 0.98$, computed from $\gamma^l = 1 - 0.16/\tau^l$. There are 100 neurons in the lower level and 50 in the higher level. We directly used the observations as input to the low level RNN. We applied truncated BPTT of length 25 for the sequential target-reaching task, as well as the consecutive relearning task.

Two separate RMSProp optimizers with decay 0.99 were used to minimize the losses of actor and critic respectively, where learning rates were 0.0003 for the critic and 0.0001 for actor. We used a replay buffer of maximum size 500,000 and performed experience replay every 2 steps, using a mini-batch containing 16 sequences with length 25, randomly sampled from the buffer (See Appendix A.2).

We summarized the hyper-parameters used in this study in Table 1. Some of the hyper-parameters were obtained by random search (see Table 1) , and the others are hand-tuned. However, a different choice of hyper-parameters will not significantly change our main conclusions (Appendix C.1).

3.2. Sequential target-reaching task

We propose a sensory-motor task, inspired by Utsunomiya and Shibata (2008), referred to as a "sequential target-reaching task". As illustrated in Fig. 2(a), a two-wheeled robot agent on a 2-dimensional field is required to reach three target positions, in a sequence, red–green–blue, without receiving any signal indicating which target to reach. The action is given by the rotation of its left and right wheels. At the beginning of each episode, the robot and targets are randomly placed on the field. The robot has sensors detecting distances and angles from the three targets and the walls (see Fig. 2(b) and Appendix A.1 for details), as well as the current-step reward. When it reaches a target in the correct sequence, it receives a one-step reward. The reward is given only if the agent followed the proper sequence, and is given only once for each target. An episode terminates if the agent completes the task or a maximum of 128 steps are taken. To successfully solve the task, the agent needs to develop the cognitive capability to remember "which target has been reached", as well as to recognize the correct sub-goal (which can be considered as approaching a target in this task).

The sequential target-reaching task is of particular interest because it abstracts many real-world tasks in complicated environments, which involve decomposition of a whole task into sub-tasks and execution of each sub-task in a specific sequence. One example is dialing on a classic telephone, where one needs to sequentially choose each number and perform detailed hand movements to dial the number. Mastering this kind of task naturally requires development of hierarchical control of actions. Lower levels acquire skills for action primitives, while higher levels learn to dispatch those action primitives in a specified sequence.

We examined ReMASTER in the sequential target-reaching task. Fig. 2(c) shows that ReMASTER can successfully solve this task through self-exploration, achieving more than 95% success rate[3] on average after training. We also tested the case in which the higher-level value function v^2 is not trained. (ReMASTER-single V in Fig. 2(c)), which achieved similar success rate in the end but the learning is relatively slower.

However, our major aim was to examine what sorts of internal representation the MTSRNN had developed for achieving sequential hierarchical control after abundant training using ReMASTER. Fig. 3(a) shows three examples of how an agent behaved after learning, where the three columns present the behavior of the same agent, but in different episodes. Interestingly, although target configurations and the motor actions (the last 2 rows of Fig. 3(b)) were completely different in these episodes, high-level neurons showed relatively similar temporal profiles of RNN outputs $\boldsymbol{c}_t^l$, as plotted in the first row of Fig. 3(a). In contrast, this feature was less obvious in the lower level (second row of Fig. 3(a)). Although we only show one agent here, this result is statistically significant (Section 3.5), and more examples can be found in Appendix Fig. C.5. This result suggests that an MTSRNN with slower dynamics in the higher level enhanced development of a consistent representation, accounting for a given sub-goal structure through abstraction in the higher level, whereas the lower level dealt with details of motor control depending on object configuration in the field in each episode. Consistency in representing sub-goals of the higher level can also be demonstrated by conducting PCA on RNN outputs of the two levels of the MTSRNN after convergence (Fig. 3(b)). We can see that the high-level RNN outputs showed a consistent, sequence-like representation of sub-goals accounted by its slower dynamics, whereas the lower level showed a more divergent representation since it needs to generate each different maneuvering trajectory. Similarly, we applied PCA on the RNN outputs, but for visualizing representation of low-level motor actions (Fig. 3(c)). It is shown that while the lower-level neurons clearly represented detailed wheel speeds, the higher-level RNN outputs were less relevant to low-level actions. Hence, we saw an emergence of action hierarchy using ReMASTER.

3.3. Consecutive relearning task

Since our previous analysis indicated that the low level learns action primitives for achieving each sub-goal, relearning to solve a new task that is a re-composition of previously learned sub-goals in a different sequence, should be much more efficient than starting from scratch.

[3] An episode was considered successful when the agent completed the task within 50 steps.

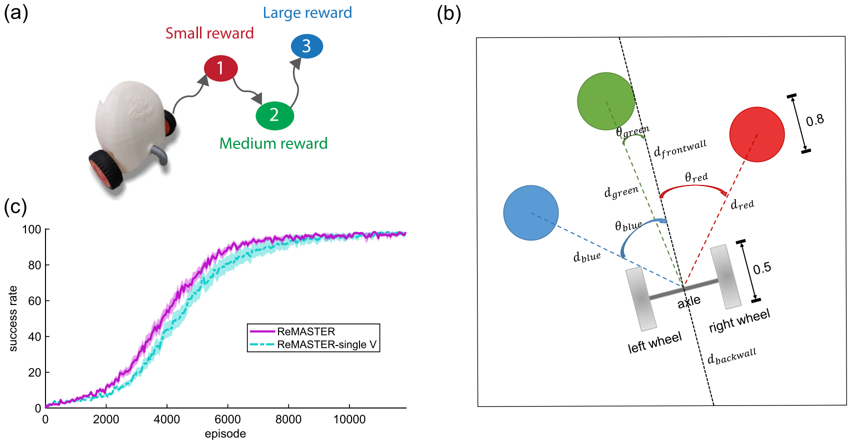

Fig. 2. The sequential target-reaching task: (**a**) Illustration of the task. Configuration of the robot and the targets was randomly initialized in each episode. (**b**) Top view of the sequential target-reaching task. The size of the square field is 15 × 15. Objects are zoomed out for visual clarity. (**c**) Performance curve indicated by success rate. ReMASTER-single V is the case in which the higher-level value function was not learned. Data are Mean ± S.E.M., obtained from 20 repeats.

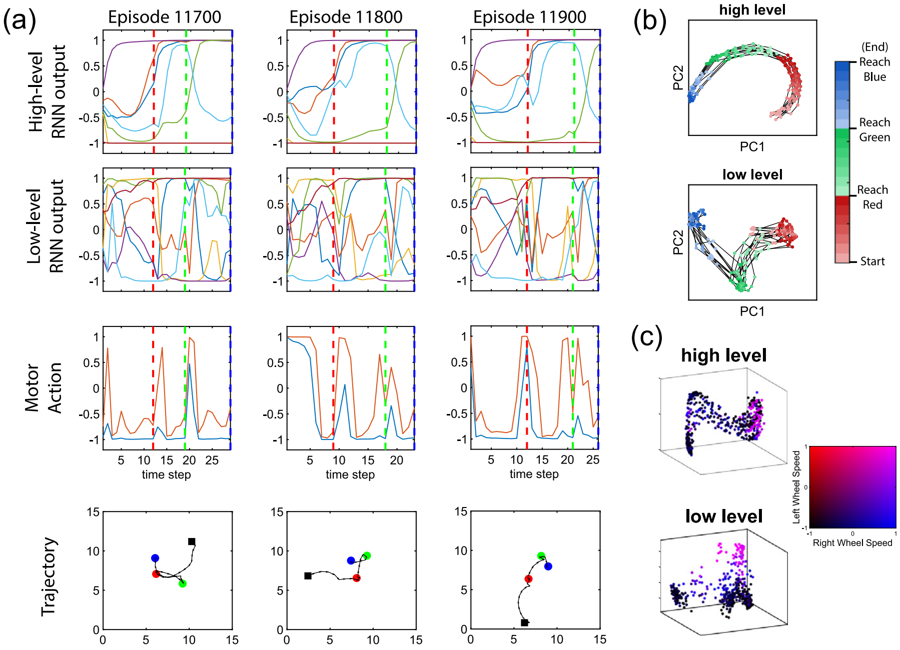

Fig. 3. Analysis on the sequential target-reaching task using ReMASTER. (**a**) Three example episodes showing the behavior of a well-trained ReMASTER agent. The first and second rows show RNN output $\mathbf{c}_t^l$ of two levels, where the vertical, dashed lines indicate the agent's reaching a target. For clarity, we plotted only c_t^l of the first 7 neurons for both levels, with different colors indicating different neurons. The motor actions indicted by velocities of the two wheels are plotted in the third row. The fourth row is the robot's trajectories, where black squares indicate its starting positions and circles are target positions. (**b**) PCA for visualizing temporal profiles of $\mathbf{c}_t^l$, using data of the same agent in (a) in episodes 11000, 11100, ..., 11900 (after convergence). Colors mean the agent is approaching to the corresponding targets, where a deeper color means the agent is more closed to the target. Samples from the same episode are linked with black lines. (**c**) Similar to (b), but the colors indicate speed of the two wheels (see the colormap). The first 3 PCs were plotted.

By considering this, we carried out another experiment, in an extended version of the sequential target-reaching task, referred to as an "consecutive relearning task" (Fig. 4(a)). In this task, the robot agent was required to adapt consecutively to changed task goals (or more specifically, changed reward functions and termination conditions) by relearning. The consecutive relearning task consisted of 3 different phases. Phase 1 corresponded to the original red–green–blue sequential target-reaching task. Phases 2 and 3 appeared as novel re-compositions of sub-goals, where the required sequences are green–blue–red and blue–green–red, respectively. While phase 1 had 12,000 episodes, there were only 3000 episodes in phase 2 or 3.

We performed experiments on this task in a lifelong learning manner (Silver, Yang, & Li, 2013; Thrun & Mitchell, 1994). We maintained the same learning algorithm and hyper-parameters throughout all 3 phases. Synaptic weights were continuously updated without resetting throughout the experiment. At the beginning of each phase, motor and neuronal noise scale were reset and the replay buffer was cleared. We additionally compared performance using ReMASTER to two alternatives, in order to examine the importance of neuronal stochasticity and intrinsic timescale hierarchy. One alternative is the deterministic version of ReMASTER in which there was only motor noise for exploration, but no noise was applied to neurons (ReMASTER-det. in Fig. 4(b–d)). Another alternative used the same algorithm, but replaced the MTSRNN with a single-layer LSTM (LSTM in Fig. 4(b-e)) using $\gamma = 1 - \sqrt{(1-\gamma^1)(1-\gamma^2)} = 0.96$, but we got similar performance for $\gamma = \gamma^1$ or γ^2). The LSTM network

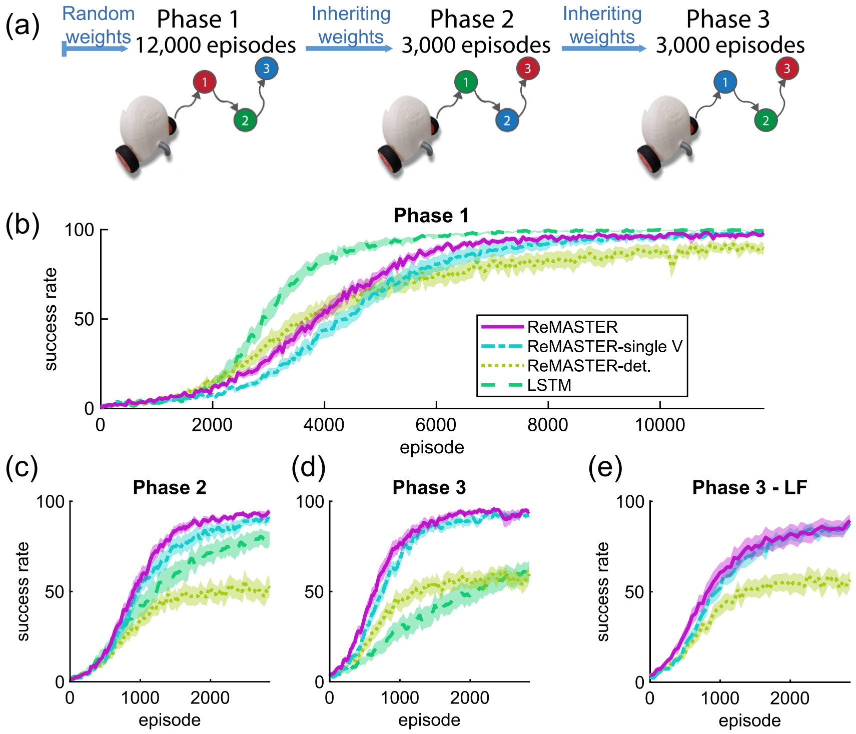

Fig. 4. The consecutive relearning task: (**a**) Illustration of the task. (**b–d**) Performance curves for all phases, plotted in the same way as Fig. 2(c). ReMASTER-det. stands for the case in which all the neurons followed deterministic dynamics, and LSTM is the alternative using the same algorithm but the network was a single-layer LSTM. (**e**) Performance curve of phase 3 with the lower-level synaptic weights frozen (Phase 3 - LF).

contained 75 cells so that the number of parameters is similar to that of the MTSRNN.

Results are illustrated in Fig. 4(b–d), which shows task performance in terms of success rate in three different phases. Several conclusions can be drawn from these results.[4]

First, for ReMASTER and ReMASTER-single V, the relearning cases of phases 2, 3 (Fig. 4(c,d)) starting with previously trained synaptic weights achieved much better sample efficiency than the case of phase 1 which was done from scratch (Note that there were only 3000 episodes in phases 2,3, whereas there were 12,000 episodes in phase 1. Rigorous comparison is left to Appendix C.2.). We consider that this resulted from compositionality during action hierarchy development, which enabled a flexible re-composition of sub-goals, so that the agents could rapidly adapt to relearning tasks.

Second, ReMASTER significantly and consistently outperformed ReMASTER-det. in all the three phases (Fig. 4(b–d)). One possible reason is that stochastic neurons could prevent the network from over-fitting, thereby enhancing network flexibility. Another is that neuronal noise can lead to larger exploration in the hidden state space (Fortunato et al., 2018; Shibata & Sakashita, 2015), which results in a greater likelihood of finding adequate neural representation in the higher level, which fits with newly appeared re-composition tasks. We also examined the cases in which neuronal noise was applied only on the higher level or the lower level, and the results are deferred to Appendix C.3.

Third, ReMASTER also addressed consistent performance advantage over ReMASTER-single V. (Fig. 4(b–d)). Recall that policy is learned to optimize the expected return with discount factor γ^1. Our results suggested it could be beneficial to learn value functions with multiple discounting, which agrees with the findings that mammalian brains are doing the same thing (Enomoto et al., 2011; Tanaka et al., 2016).

Finally, ReMASTER and ReMASTER-single V showed a performance advantage over the LSTM alternative in phases 2, 3, although LSTM achieved great performance in phase 1 (Fig. 4(b–d)). We consider the performance degradation of LSTM in phases 2,3 is because of the mixed representation of sub-goal sequencing and detailed motor skills in one level. This created difficulty in relearning sub-goal sequencing while reusing low-level skills. In contrast, ReMASTER provided flexible compositionality that enables these two levels of control to be better segregated in different levels in MTSRNN. Although biological plausibility of our approach is arguable, this result may underlie a potential reason of why we have many separated, timescale-distinct brain regions working for multiple levels of functions (Murray et al., 2014; Runyan et al., 2017; Wang et al., 2018).

3.4. *Learning to solve new tasks with low-level weights frozen*

The previous results (Fig. 4(b–d)) were obtained when both the higher and the lower level synaptic weights were continually trained throughout the task. However, if the lower level had acquired necessary motor skills for achieving the sub-goals, it should be possible for the agent to learn to solve new tasks by updating only the higher level.

Therefore, we conducted another simulation on the consecutive relearning task using ReMASTER, in which low-level synaptic weights (purple connections in Fig. 1) were frozen in phase 3, as inherited at the end of phase 2. The ReMASTER and ReMASTER-single V agents showed remarkable learning effectiveness in phase 3 (Fig. 4(e)), whereas the ReMASTER-det. agents could improve their policy but the learning was less efficient. This finding further supports our speculation that hierarchical action control had developed in phases 1 and 2, wherein motor skills for

[4] Although we used tuned hyperparameters for better performance, these conclusions indeed hold for different choice of hyperparameters (Appendix C.1).

Table 2
Consistency of RNN outputs in representing sub-goals among the last 1000 episodes in each phase. Data are Mean ± STD.

Network	Phase 1	Phase 2	Phase 3
ReMASTER (high level)	**0.95** ± **0.02**	**0.94** ± **0.03**	**0.95** ± **0.02**
ReMASTER (low level)	0.81 ± 0.04	0.80 ± 0.05	0.80 ± 0.05
ReMASTER-single V (high level)	0.88 ± 0.06	0.86 ± 0.06	0.86 ± 0.06
ReMASTER-single V (low level)	0.77 ± 0.06	0.80 ± 0.04	0.82 ± 0.04
ReMASTER-det. (high level)	0.88 ± 0.04	0.79 ± 0.06	0.85 ± 0.04
ReMASTER-det. (low level)	0.75 ± 0.05	0.64 ± 0.07	0.71 ± 0.04
LSTM	**0.85** ± **0.20**	0.78 ± 0.20	0.54 ± 0.29

achieving sub-goals had developed in the low-level and memory for sequencing sub-goals had developed in the high-level, and this was facilitated by neuronal noise.

3.5. Consistency in representing sub-goals

To understand the underlying neural mechanisms for ReMASTER's promising performance in relearning phases (Fig. 4(c,d)), we analyzed neural data by looking at how consistent the RNN outputs of different RNN architectures could represent sub-goals in each phase.

We measured consistency in representing sub-goals by cosine similarity of temporal profile of the RNN outputs $\boldsymbol{c}^l$ across the last 1000 episodes of each phase (see Appendix B.1 for details), for both of the higher level and the lower one (Table 2). It can be seen that higher consistency mostly corresponds to higher success rate for the three models in the consecutive relearning task (Fig. 4(b–d)), where ReMASTER agents always showed great consistency in representing sub-goals in the higher level, in contrast to the alternatives, the performance and consistency of which decreased significantly in later phases. We do not show here the comparison across different phases because the RNN outputs corresponding to the sub-goals could be different when an agent adapts to a new phase (see Appendix C.4 for more discussion). However, it is rather important that higher flexibility for re-organizing sub-goal representation was shown using ReMASTER agents.

3.6. Manipulating agent behaviors by clamping high-level neural states

For animals, different brain regions often serve at different levels in action generation. For instance, premotor areas of rodent motor cortex are thought to be important in action choices, while primary motor cortex is considered responsible for details in action execution (Morandell & Huber, 2017). More interestingly, experimental studies have demonstrated that action primitives of animals can be altered by electrophysiological stimulation or optogenetic inactivation to certain upstream neurons (Morandell & Huber, 2017; Vu, Mazurek, & Kuo, 1994).

Here, we consider analogous experiments on artifacts with ReMASTER agents. We first randomly picked an agent after finishing the consecutive relearning task and then computed the average of $\boldsymbol{c}^2$ and $\boldsymbol{u}^2$ over the last 500 episodes of phase 3, at the middle step of (i) from initial position to the blue target; (ii) from the blue target to the green one; (iii) from the green target to the red one. By clamping high-level RNN states ($\boldsymbol{c}^2$, $\boldsymbol{u}^2$) to those of (i), (ii), or (iii), we could "manipulate" a trained agent to consistently follow an action primitive pursuing the corresponding sub-goal (Fig. 5). In contrast, fixing low-level RNN states only results in a constant (noisy) action, which is directly determined by $\boldsymbol{c}^1$. Therefore, the high-level RNN states act as a label for the action primitives. The continuous property of the RNN states enables representation of an arbitrary number of sub-goals, where in our case we can readily find 3 meaningful action primitives corresponding to the 3 targets.

3.7. Timescales and discountings

We have been discussing the role of multiple timescales, indicated by τ^l, the time constant of the lth-level RNN, and γ^l, the discount factor of the lth-level value function. In our experiments using ReMASTER, the lower level had smaller τ^1(=2) and γ^1(=0.92), corresponding to a fast dynamic, whereas the higher level was characterized by a slower timescale ($\tau^2 = 8, \gamma^2 = 0.98$). However, a computational validation of this "the-higher-the-slower" setting should also be conducted.

For this purpose, different settings of τ^l and γ^l were examined in the consecutive relearning task. The simulation results (Fig. 6) demonstrated a clear advantage of the setting we used, compared to other cases in which "the-higher-the-slower" was not followed. Exchange of values of γ^1 and γ^2 resulted in significant performance degradation, while alternating values of τ^1 and τ^2 showed even worse performance. Also, it appeared as an unsatisfying choice to set medium values of τ and γ for both layers. The results suggested that a "the-higher-the-slower" setting in our model corresponds should be adopted for better performance, which agrees with neurobiological experiments that the higher-level brain regions have longer intrinsic timescales (Murray et al., 2014).

4. Related work

Despite much early effort spent on hierarchical RL (Dietterich, 2000; Sutton et al., 1999) using pre-defined action hierarchies, a number of recent studies have been focused on discovering action primitives[5] that serve for hierarchical RL. More recent works (Bacon et al., 2017; Brunskill & Li, 2014; Fox et al., 2017; Riemer, Liu, & Tesauro, 2018) were extensions of the *option* framework (Sutton et al., 1999), which introduced a termination variable to determined start and end of an action primitive. Their works required a pre-defined number of options, whereas our framework can learn to represent an arbitrary number of options by high-level RNN states (Fig. 5). Moreover, most of these studies focused on tasks that do not require long-term credit assignment or memorization. In this paper, we consider a different scheme wherein the agent needs to accomplish a series of sub-goals in a particular sequence without observable information indicating the current sub-goal. Such a scheme is common in real life, but has rarely been investigated in RL.

Another track of related studies is skill sharing/reuse among similar tasks in RL. Studies have been conducted considering various schemes, such as meta-RL (Al-Shedivat et al., 2018; Finn, Abbeel, & Levine, 2017; Finn, Xu, & Levine, 2018; Santoro, Bartunov, Botvinick, Wierstra, & Lillicrap, 2016; Wang et al., 2018; Yoon et al., 2018) and lifelong RL (Abel, Jinnai, Guo, Konidaris, & Littman, 2018; Mankowitz, Mann, & Mannor, 2016; Rusu et al., 2016; Silver et al., 2013; Tessler, Givony, Zahavy, Mankowitz, & Mannor, 2017). In particular, some authors proposed ideas shared with our work. Several studies (Al-Shedivat et al., 2018; Santoro et al., 2016; Wang et al., 2018) employed RNNs for their meta-learning competency, and the others (Brunskill & Li, 2014; Tessler et al., 2017) suggested that reuse of action primitives can enhance lifelong learning. However, many of these works considered a multi-task setting where the agent repetitively interacts with a random task sampled from a task set. In our case, the agent first learned to solve one task (phase 1), and the self-developed action hierarchy facilitated relearning in new tasks (phases 2 and 3) with which the agents had never interacted.

[5] "Action primitive" in our paper has a similar meaning as *option* (Sutton et al., 1999) in some related literature (Bacon et al., 2017, 2017; Brunskill & Li, 2014; Fox, Krishnan, Stoica, & Goldberg, 2017).

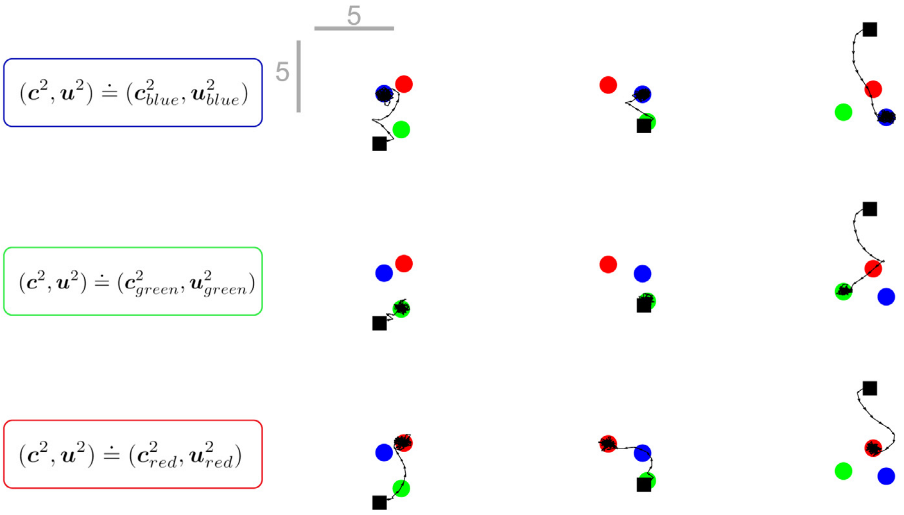
5
5
$(c^2, u^2) \doteq (c^2_{blue}, u^2_{blue})$
$(c^2, u^2) \doteq (c^2_{green}, u^2_{green})$
$(c^2, u^2) \doteq (c^2_{red}, u^2_{red})$

Fig. 5. Manipulating agent behaviors by clamping high-level RNN states. All trajectories were from one agent and each row used same high-level RNN states. Black squares and colored circles indicate the agent's initial positions and the target positions, respectively. Each column used the same random seeds for generating initial position and target positions.

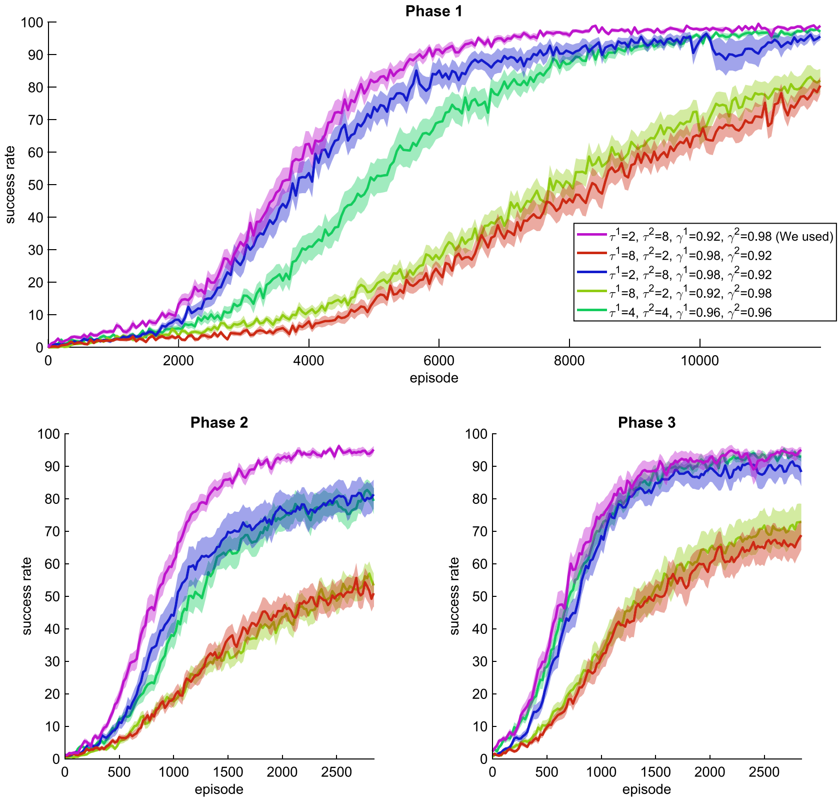

Fig. 6. Performance comparison among different settings of τ^l and γ^l. Each result was obtained from 10 repeats.

5. Conclusion and future work

In the current study, we focused on a type of sequential compositional task and comprehensively investigated how they can be solved by autonomously developing sub-goal structure with acquiring necessary action primitives via RL. For this purpose, we proposed a novel RL framework, ReMASTER, which is characterized by two essential features. One is the multiple timescale property both in neural activation dynamics and reward discounting, which is inspired by neuroscientific findings (Huys et al., 2004; Murray et al., 2014; Newell et al., 2001; Runyan et al., 2017; Smith et al., 2006). The other is stochasticity introduced in neural units in all layers, also inspired by the corresponding biological facts (Beck et al., 2008, 2012; Orbán, Berkes, Fiser, & Lengyel, 2016).

Simulation results showed that action hierarchy is emerged by developing an adequate internal neuronal representation at multiple levels. We presented several pieces of evidences showing that compositionality developed in the network by taking advantage of multiple timescales: abstract action control in terms of sequencing of sub-goals developed in the higher level, while a set of action primitives as skills for detailed sensory-motor control for achieving each sub-goal acquired in the lower level.

Furthermore, compositionality developed in the previous learning enabled efficient relearning in adaptation to changed task goals that involved re-composition of previously learned sub-goals. This re-composition capability was further enhanced with introduction of neuronal noise in addition to motor noise. Such adaptation became possible because development of hierarchical control using multiple levels allowed enough flexibility for re-composition of previously learned control skills.

Since our experiments showed that exchange of timescales between the higher and lower levels resulted in significant performance degradation, it should be worth investigating how an optimal timescale for each level can be determined autonomously during task execution. One possibility is to incorporate LSTMs cascaded in multiple levels (Vezhnevets et al., 2017) with the expectation that the forgetting gate in LSTMs could provide the means for adaptive timescale modulation.

Finally, ReMASTER is flexible in adopting any gradient-based actor–critic algorithms. Performance can be further improved by employing well-designed model-free algorithms such as (Haarnoja et al., 2018; Wang et al., 2017). Also, Recent model-based RL methods have addressed promising performance using probabilistic state transition models (Deisenroth & Rasmussen, 2011; Ha & Schmidhuber, 2018; Kaiser et al., 2019). In this respect, it should be interesting to combine ReMASTER with probabilistic inference of state transitions, using, e.g., a multiple timescale Bayesian RNN (Ahmadi & Tani, 2019; Chung et al., 2015). Such trials will be attempted in future studies.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Japan Society for the Promotion of Science KAKENHI Grant Numbers JP16K21738, JP16H06561 and JP16H06563 to KD and 17H06034 to JT, and research support of Okinawa Institute of Science and Technology, Japan Graduate University to KD and JT. We would like to thank Tadashi Kozuno and Siqing Hou for insightful discussions, and Steven Aird for helping improving the manuscript.

Appendix A. Task settings

A.1. Sequential target-reaching task

The setting for a sequential target-reaching task is similar to that in Utsunomiya and Shibata (2008). A two-wheel robot is required to approach three targets in a sequence, on a 2-dimensional field, as showed in Fig. 2(b). The 2-D field is a 15 × 15 square area, restrained by walls. The robot agent has two wheels of radius 0.25, connected by an axle. It receives sensory signals to detect distances and angles to the target as well as the walls, as shown in Fig. 2(b). At each step, the action is given as the rotations of two wheels, which are continuous in range $[-180°, 180°]$. Length of the axle is 1, so that the robot can turn 90° at most in one step. There are three targets, indicated by red, green and blue, each of which is a circular area of radius 0.4. At the beginning of each episode, positions of three targets are randomly set inside the center 8 × 8 area. The distance between two targets are ensured to be larger than 2. The observation is a 12-D real number vector: $(e^{-d_{red}/5}, e^{-d_{green}/5}, e^{-d_{blue}/5}, e^{-d_{front\,wall}/5}, e^{-d_{backwall}/5}, r, \sin\theta_{red}, \cos\theta_{red}, \sin\theta_{green}, \cos\theta_{green}, \sin\theta_{blue}, \cos\theta_{blue})$, where r is the immediate reward at current time step, and other quantities are shown in Fig. 2(b).

The robot must reach the three targets in the sequence red–green–blue to maximize rewards. The reward function is given as:

If $d_{red}(\tau) > 0.4\,\forall\,\tau < t$ and $d_{red}(t) \leq 0.4$, then

$$r(t) = 0.8/(1 + d_{red}(t)). \tag{8}$$

If $\exists\,\tau < t$ that $d_{red}(\tau) \leq 0.4$, and $d_{green}(\tau) > 0.4\,\forall\,\tau < t$, $d_{green}(t) \leq 0.4$, then

$$r(t) = 2.0/(1 + d_{green}(t)). \tag{9}$$

If $\exists\,\tau < t$ that $d_{red}(\tau) \leq 0.4$, and $\exists\,\tau < t$ that $d_{green}(\tau) \leq 0.4$, and $d_{blue}(t) \leq 0.4$, then

$$r(t) = 5.0/(1 + d_{green}(t)) \quad \text{(task done)}. \tag{10}$$

If the robot hits the walls, a negative reward −0.1 is given. Otherwise the reward is zero.

A.2. Replay buffer for dispersed replay

To enable experience replay, we stored state transitions $(s_t, s_{t+1}, a_t, r_t, done_t)$ and RNN states $(\boldsymbol{c}_t, \boldsymbol{\mu}_t)$ in a replay buffer. We also recorded behavior policy π_t in it to compute the importance sampling ratio. We did not separate episodes in the replay buffer. Instead, we consecutively recorded every step, and padded $L - 1$ steps at which gradients were not calculated, when an episode terminated. Then, we could randomly sample n sequences of length-l as a minibatch for truncated BPTT, with sampling bias.

A.3. Initial RNN states for experience replay

Different from feedforward neural networks, RNNs for off-policy RL have some practical problems. One major problem is how to decide initial states when training a sequence sampled from the replay buffer. When dealing with finite horizon (episodic) RL tasks, applicable approaches can be summarized as:

- **Recording the RNN states at each step.** RNN states can be treated as hidden observations used in training, which need to be recorded in the replay buffer. Despite the simplicity of this approach, it is unclear what algorithmic issues will be introduced by difference between old internal representations and new ones.
- **Using an entire episode as a sequence.** This was used, e.g., in Mnih et al. (2016), providing zero initial states for all the episodes. However, this implementation is computationally inefficient when the length of some episodes is large.
- **Using random sequences with zero initial states.** Sample sequences are randomly sampled from the entire memory, given all-zero initial states. This approach was used in Hausknecht and Stone (2015) for experiments in Atari Games. Unfortunately, this implementation prevents learning long-term dependence because of the mismatch of initial states, as argued in Kapturowski et al. (2018).
- **Replaying the sequences.** Starting with zero initial states at each episode, RNN states for off-policy updates can be obtained by unrolling new RNNs on old trajectories. A modified version of this approach is offered in Kapturowski et al. (2018), where the authors assume that computing the forward dynamic of RNNs can help them find better RNN states from zero or recorded RNN states, starting e.g., 20 steps

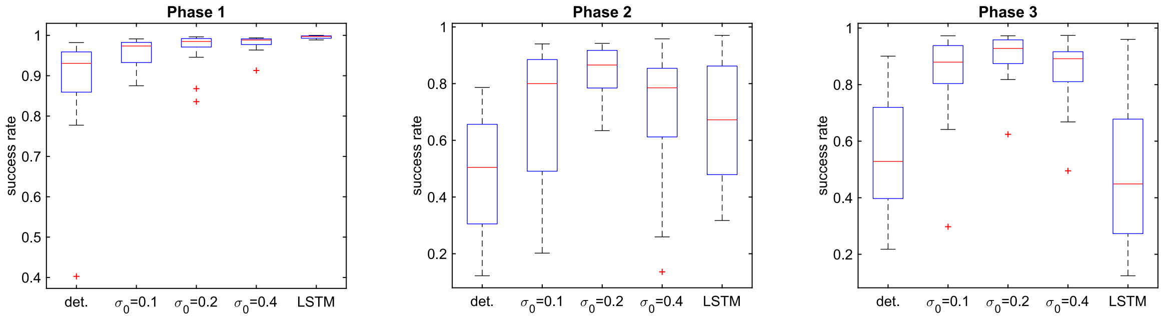

Fig. A.1. Final performances of ReMASTER for different scales of neuronal noise obtained from last 1000 episodes of each phase.

before the start of a sampled sequence. Although Kapturowski et al. (2018) demonstrated remarkable performance on many RL tasks using this approach, their assumption has not been systematically discussed.

For simplicity, we employ the first approach. Our experimental results show that it is practical. However, how to choose better initial states still remains a challenge for RL with experience replay using RNNs.

A.4. Noise scale

For continuous sensory-motor tasks, the range of state space can be very large. To enhance efficiency of motor exploration, we used motor noise generated by the *Ornstern–Uhlenbeck process* (Uhlenbeck & Ornstein, 1930) (OU-process), like that used in Lillicrap et al. (2015). The OU-process generates temporally autocorrelated noise; thus, the exploration range can be increased with the "inertia" of the noise. However, it is not necessary to apply temporally correlated noise to hidden states of the MTSRNN, since recurrent connections in an RNN autonomously generate it.

For exploration in all the tasks, we applied auto-correlated Gaussian noises to the robot's actions, which were generated by independent OU-processes. Each action noise x_t can be computed by

$$x_t = -\theta_a x_{t-1} + e\sqrt{2\theta_a}\epsilon_t, \tag{11}$$

where $\theta_a = 0.3$ for all of our experiments, and ϵ_t is a unit of Gaussian white noise. e indicates the scale of action noise, which was annealed exponentially w.r.t. episodes, with a minimum value of 0.1:

$$e = 180^\circ \times \left[0.75 \times \exp(-\frac{1}{3000} \times \text{episode}) + 0.1\right]. \tag{12}$$

Meanwhile, neuronal stochasticity is given by Gaussian white noise with scale

$$\sigma = \sigma_0 \exp(-\frac{1}{3000} \times \text{episode}). \tag{13}$$

We performed experiments to determine the proper value of σ_0, and found that $\sigma_0 = 0.2$ gave rise to better performance (Fig. A.1). Thus we used $\sigma_0 = 0.2$.

For the consecutive relearning task, at the beginning of phases 2 and 3, we cleared the memory buffer and reset the noise scale, annealed as

$$e = 180^\circ \times \left[0.75 \times \exp(-\frac{1}{750} \times \text{episode}) + 0.1\right], \tag{14}$$

and

$$\sigma = \sigma_0 \exp(-\frac{1}{750} \times \text{episode}). \tag{15}$$

Appendix B. Data analysis

B.1. Consistency in representing sub-goals

This section describes how we computed consistency in representing sub-goals (Section 3.5) by cosine similarity of the RNN outputs $\boldsymbol{c}$ across different episodes. Because there are usually different numbers of time steps in each episode, we first normalized the number of time steps to 30 for all successful episodes. The normalized time step $t_{norm} = 1$ when the agent starts an episode, and $t_{norm} = 10, 20$ and 30 when the agent reaches the first, second, and third target, respectively. Then $\boldsymbol{c}^l_{t_{norm}}$ can be obtained w.r.t. the normalized time steps by linear interpolation. Therefore, if the higher-level RNN outputs can consistently represent the correct sub-goals, their temporal profiles w.r.t. normalized time steps $\boldsymbol{c}^2_{t_{norm}}$ should be similar among different episodes e after convergence.

Then cosine similarity of $\boldsymbol{c}^l_{t_{norm}}$ was then computed for each agent by

$$\text{Consistency}^l = \text{Mean}_{e_i \neq e_j, t_{norm}, k}\left[\text{CosSim}\left(c^l_{e_i,k,t_{norm}}, c^l_{e_j,k,t_{norm}}\right)\right], \tag{16}$$

where $\boldsymbol{c}^l_{e,k,t_{norm}}$ indicates the temporal profile (using the normalized time step t_{norm}) of the RNN output of the kth neuron of the lth level in the eth episode. e_i and e_j are successful episodes in the last 1000 episodes of each phase.

Appendix C. Supplementary results

C.1. Effect of hyperparameters

Many RL algorithms suffer from a proper choice of hyperparameters (such as learning rate, number of neurons in the network) in terms of a satisfying performance. It is also important for us to make sure that the our main results are robust to hyperparameters. For this purpose, we did a random search for hyperparameters (Fig. C.1). More specifically, the sequence length for BPTT was sampled log-uniformly in [10, 40]. The learning rate for the actor and for the critic was sampled log-uniformly in [0.00015, 0.0006] and [0.00005, 0.0002], respectively. For the MTSRNN, the number of neurons was log-uniformly in [25, 100] in the lower level, and [50, 200] in the higher-level. The number of LSTM cells was in [40, 160], also log-uniformly sampled.

As shown in Fig. C.1, although the overall performance was a little worse than that using tuned hyperparameters, our conclusions in Section 3.3 did not vary. The LSTM alternative performed better in phase 1, but became worse in the latter phases. Also,

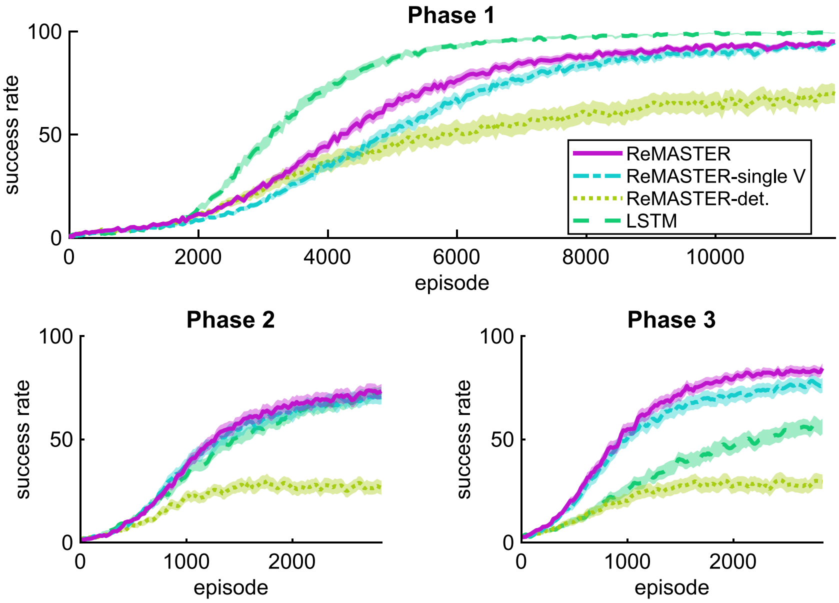

Fig. C.1. Performance in the consecutive relearning task, using a range of hyperparameters.

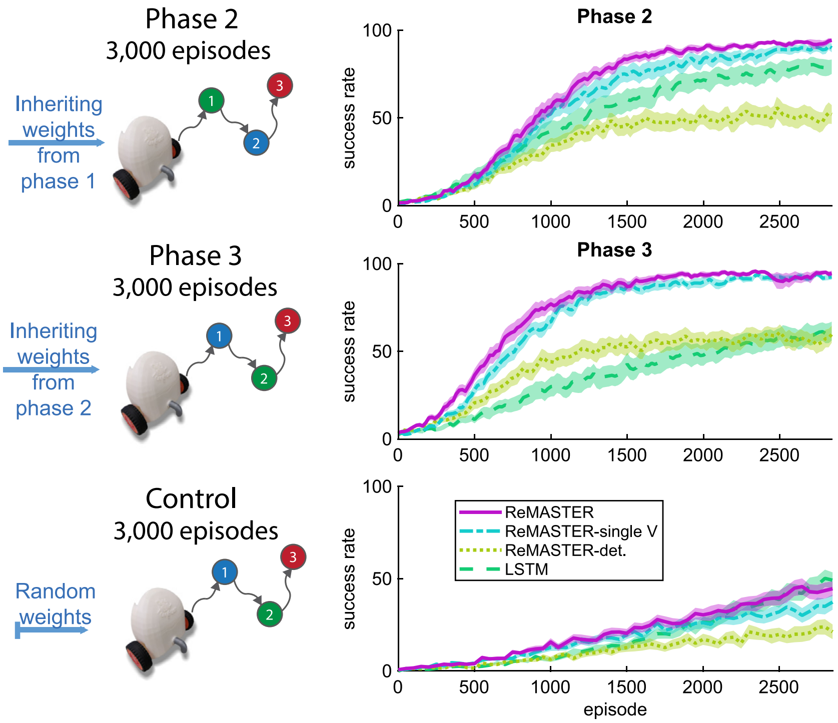

Fig. C.2. Performance comparison among phases 2, 3 and the control case.

ReMASTER always outperformed ReMASTER-det. and ReMASTER-single V.

C.2. *Performance gain with inherited weights*

We prepared a control task that is equal to phase 3 (also equivalent to phase 2 because of symmetry of the three targets) except a random initialization of synaptic weights at the beginning (Fig. C.2, Bottom). It can be seen that agents with inherited weights largely outperformed agents in the control case that start from scratch, showing meta-learning competency of RNNs (Wang et al., 2018).

C.3. *Neuronal noise ablation study*

We further conducted experiments to investigate the role of neuronal noise in either the higher level and the lower level. The results (Fig. C.3) show that, lack of neuronal noise in the higher level leads to slightly worse performance in relearning phases. When the lower-level neuronal followed deterministic dynamics, although it learned slightly faster in phase 1, significant performance degradation was observed in phases 2 and 3. Also, lack of the higher-level stochasticity lead to slightly worse performance in all 3 phases.

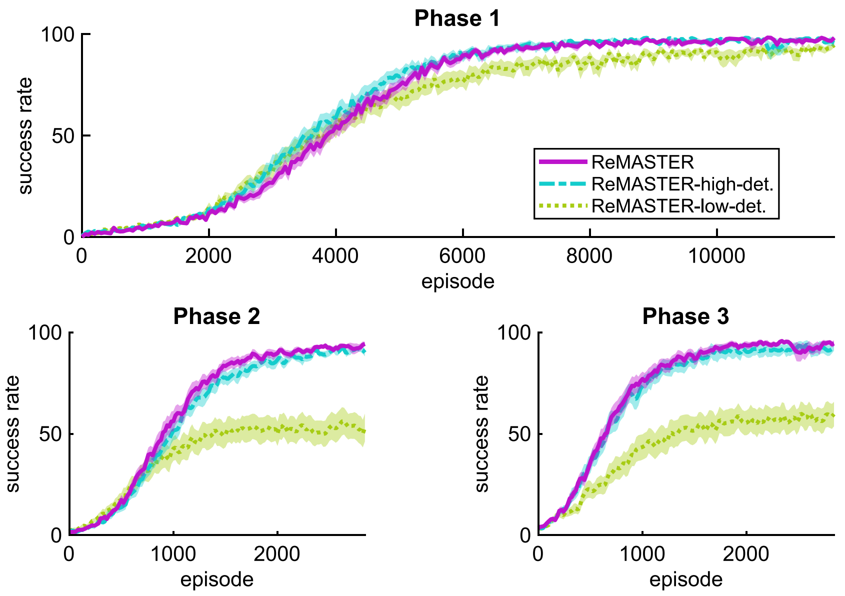

Fig. C.3. Success rate in all 3 phases. ReMASTER is compared to ReMASTER-high-det. and ReMASTER-low-det., in which the higher-level or the lower-level neurons are deterministic.

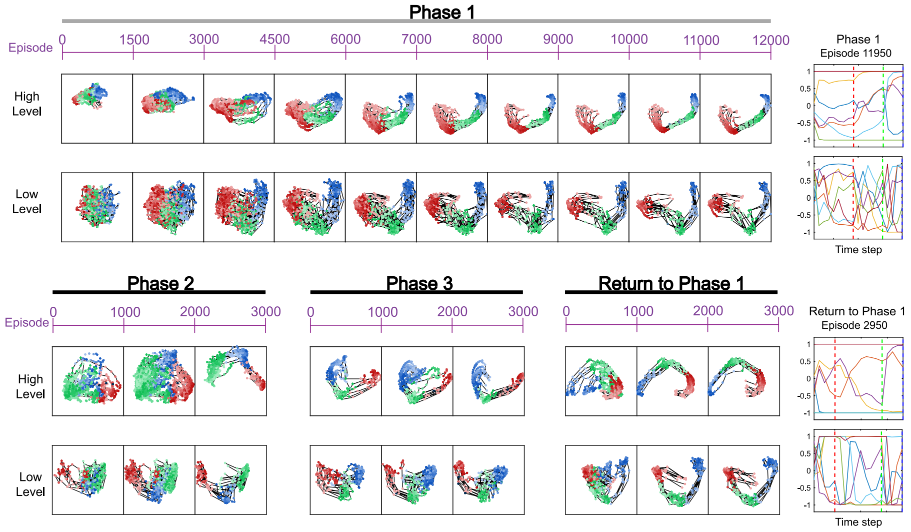

Fig. C.4. Development of internal representation for sub-goals. Left: The first two PCs of internal representations of an ReMASTER agent, after performing PCA for each phase separately, plotted in the same way as Fig. 3(b). Right: Profiles of RNN outputs of the higher level (Top) and the lower level (Bottom), plotted in the same way as Fig. 3(a).

C.4. Development of internal representations

To see how internal representation of ReMASTER agents develops throughout learning, we performed PCA for the RNN outputs in different periods of learning and visualized the first 2 PCs[6] of a randomly selected ReMASTER agent (Fig. C.4, Left). The agent was trained to learn the consecutive relearning tasks with 3 phases. In addition, we let the agent consecutively adapt to the fourth phase wherein the task goal was the same as in phase 1 ("Return to Phase 1" in Fig. C.4) to see how the representation varies for the same task goal but at different learning stages.

In consistency with the result that learning was faster in later phases (Fig. 4(b,c)), internal representation also converged faster in relearning phases (Fig. C.4, Left). In particular, when the agent returned to phase 1, it only took less than 1000 episodes to achieve a converged representation of sub-goals (The first 2 PCs after episode 1000 are almost invariant).

Also, it can be seen that the internal representation in phase 1 after convergence is different from that when the agent returns to phase 1 (Fig. C.4, Right). This can also be demonstrated using the similarity measure (Appendix B.1) between the RNN outputs in these two phases (averaged from last 1000 episode in each phase), which is 0.37 ± 0.07 for the lower level and 0.33 ± 0.09 for the higher level, on 20 trials.

However, the first 2 PCs interestingly show that the representations at the end of phase 1 and phase 4 have similar structure, despite that the basis vectors are different (Fig. C.4, Left). To demonstrate this, we performed linear transformation (stretching, rotation and reflection) to the first 2 PCs to maximize their similarity between the two phases (again, averaged from last 1000 episode in each phase). The result showed a similarity

[6] Note that PCA was conducted for each phase separately. This is because sub-goals representations were re-organized when adapting to a new phase, and thus we failed to obtain clear visualization of sub-goal representations across phases.

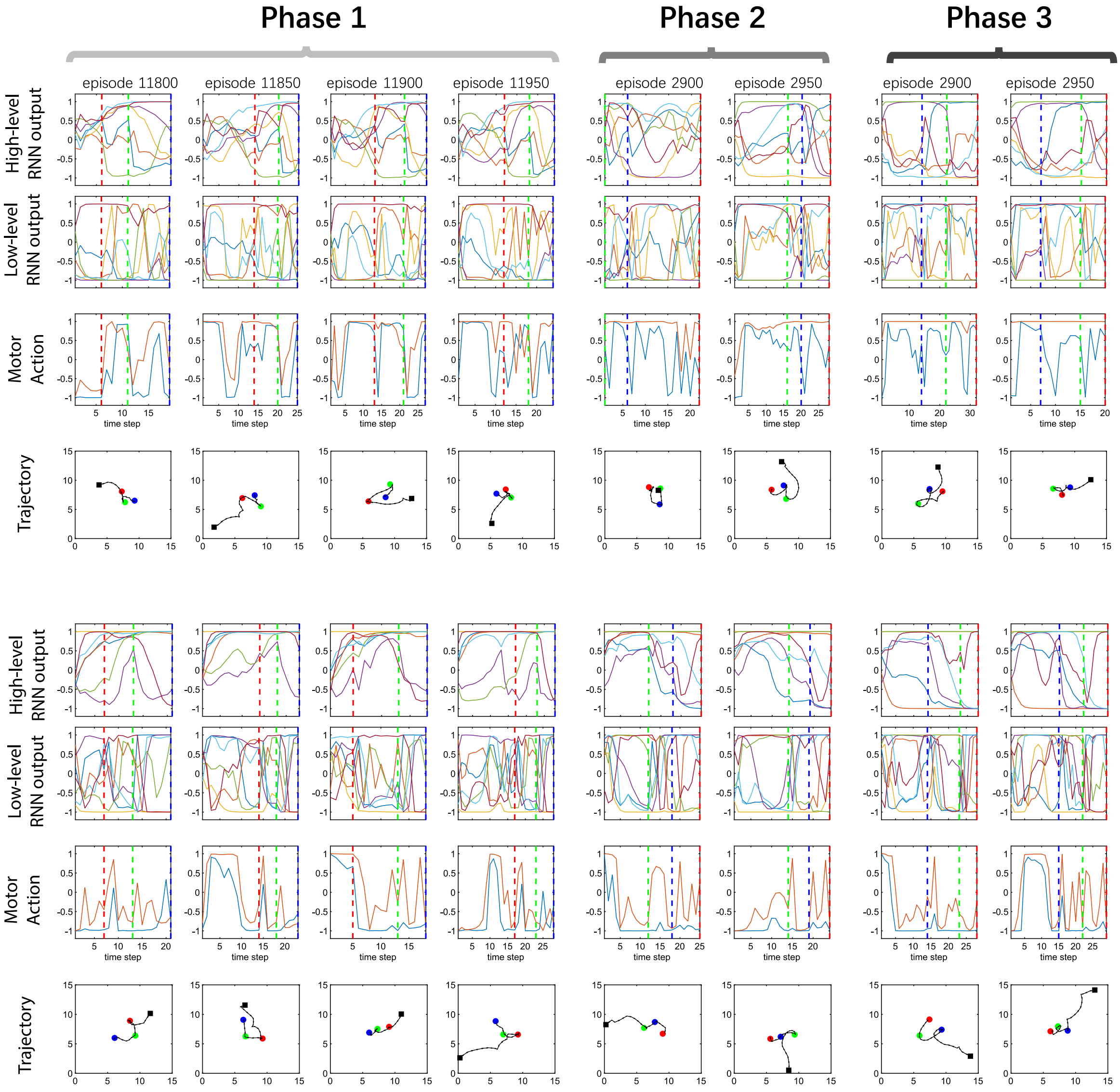

Fig. C.5. Example episodes showing the behavior of two well-trained ReMASTER agents, in all 3 phases. Plotted in the same way as Fig. 3(a) in the main paper. For clarity, the first 7 neurons are plotted for both levels, with different colors indicating different neurons.

of 0.93 ± 0.07 for the lower level and 0.97 ± 0.03 for the higher level on 20 trials, which is much higher than that of RNN outputs. This suggests that a robust sub-goal encoding scheme was achieved in the proposed model either by sufficient amount of learning from scratch (phase 1) or by relearning, in which the internal representation was acquired much more quickly.

References

Abel, D., Jinnai, Y., Guo, S. Y., Konidaris, G., & Littman, M. (2018). Policy and value transfer in lifelong reinforcement learning. In *Proceedings of the 35th international conference on machine learning* (pp. 20–29).

Ahmadi, A., & Tani, J. (2019). A novel predictive-coding-inspired variational RNN model for online prediction and recognition. *Neural Computation*, 1–50.

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., & Abbeel, P. (2018). Continuous adaptation via meta-learning in nonstationary and competitive environments. In *Proceedings of the international conference on learning representations*.

Åström, K. J. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, *10*(1), 174–205.

Bacon, P.-L., Harb, J., & Precup, D. (2017). The option-critic architecture. In *Thirty-First AAAI conference on artificial intelligence*.

Badre, D., & Frank, M. J. (2011). Mechanisms of hierarchical reinforcement learning in cortico–striatal circuits 2: Evidence from fmri. *Cerebral Cortex*, *22*(3), 527–536.

Badre, D., Kayser, A. S., & D'Esposito, M. (2010). Frontal cortex and the discovery of abstract action rules. *Neuron*, *66*(2), 315–326.

Beck, J. M., Ma, W. J., Kiani, R., Hanks, T., Churchland, A. K., Roitman, J., et al. (2008). Probabilistic population codes for Bayesian decision making. *Neuron*, *60*(6), 1142–1152.

Beck, J. M., Ma, W. J., Pitkow, X., Latham, P. E., & Pouget, A. (2012). Not noisy, just wrong: the role of suboptimal inference in behavioral variability. *Neuron*, *74*(1), 30–39.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 679–684.

Brunskill, E., & Li, L. (2014). PAC-inspired option discovery in lifelong reinforcement learning. In *Proceedings of the 31th international conference on machine learning* (pp. 316–324).

Chaudhuri, R., Knoblauch, K., Gariel, M.-A., Kennedy, H., & Wang, X.-J. (2015). A large-scale circuit mechanism for hierarchical dynamical processing in the primate cortex. *Neuron*, *88*(2), 419–431.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., & Bengio, Y. (2015). A recurrent latent variable model for sequential data. In *Advances in neural information processing systems* (pp. 2980–2988).

Degris, T., White, M., & Sutton, R. S. Off-policy actor-critic. In *Proceedings of the 29th international conference on machine learning* (pp. 179–186).

Deisenroth, M., & Rasmussen, C. E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th international conference on machine learning* (pp. 465–472).

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, *12*(1), 219–245.

Enomoto, K., Matsumoto, N., Nakai, S., Satoh, T., Sato, T. K., Ueda, Y., et al. (2011). Dopamine neurons learn to encode the long-term value of multiple future rewards. *Proceedings of the National Academy of Sciences*, 201014457.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th international conference on machine learning* (pp. 1126–1135).

Finn, C., Xu, K., & Levine, S. (2018). Probabilistic model-agnostic meta-learning. In *Advances in neural information processing systems* (pp. 9516–9527).

Florensa, C., Duan, Y., & Abbeel, P. (2017). Stochastic neural networks for hierarchical reinforcement learning. In *Proceedings of the international conference on learning representations*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., et al. (2018). Noisy networks for exploration. In *Proceedings of the international conference on learning representations*.

Fox, R., Krishnan, S., Stoica, I., & Goldberg, K. (2017). Multi-level discovery of deep options. arXiv preprint arXiv:1703.08294.

Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016). Sequential neural models with stochastic layers. In *Advances in neural information processing systems* (pp. 2199–2207).

Frans, K., Ho, J., Chen, X., Abbeel, P., & Schulman, J. (2018). Meta learning shared hierarchies. In *Proceedings of the international conference on learning representations*.

Friedman, J. H. (1997). On bias, variance, 0/1–loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, *1*(1), 55–77.

Ha, D., & Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (pp. 2450–2462).

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th international conference on machine learning* (pp. 1856–1865).

Hartmann, C., Lazar, A., Nessler, B., & Triesch, J. (2015). Where's the noise? Key features of spontaneous activity and neural variability arise through learning in a deterministic network. *PLoS Computational Biology*, *11*(12), e1004640.

Hausknecht, M., & Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*.

Heess, N., Hunt, J. J., Lillicrap, T. P., & Silver, D. (2015). Memory-based control with recurrent neural networks. arXiv preprint arXiv:1512.04455.

Huys, R., Daffertshofer, A., & Beek, P. J. (2004). Multiple time scales and multiform dynamics in learning to juggle. *Motor Control*, *8*(2), 188–212.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., et al. (2019). Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, *364*(6443), 859–865.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., et al. (2019). Model-based reinforcement learning for Atari. arXiv preprint arXiv:1903.00374.

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., & Munos, R. (2018). Recurrentexperience replay in distributed reinforcement learning. In *Proceedings of the international conference on learning representations*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Mankowitz, D. J., Mann, T. A., & Mannor, S. (2016). Adaptive skills adaptive partitions. In *Advances in neural information processing systems* (pp. 1588–1596).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33th international conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Morandell, K., & Huber, D. (2017). The role of forelimb motor cortex areas in goal directed action in mice. *Scientific Reports*, *7*(1), 15759.

Murray, J. D., Bernacchia, A., Freedman, D. J., Romo, R., Wallis, J. D., Cai, X., et al. (2014). A hierarchy of intrinsic timescales across primate cortex. *Nature Neuroscience*, *17*(12), 1661.

Newell, K. M., Liu, Y.-T., & Mayer-Kress, G. (2001). Time scales in motor learning and development. *Psychological Review*, *108*(1), 57.

Orbán, G., Berkes, P., Fiser, J., & Lengyel, M. (2016). Neural variability and sampling-based probabilistic representations in the visual cortex. *Neuron*, *92*(2), 530–543.

Ramirez, S., Liu, X., Lin, P.-A., Suh, J., Pignatelli, M., Redondo, R. L., et al. (2013). Creating a false memory in the hippocampus. *Science*, *341*(6144), 387–391.

Riemer, M., Liu, M., & Tesauro, G. (2018). Learning abstract options. In *Advances in neural information processing systems* (pp. 10424–10434).

Runyan, C. A., Piasini, E., Panzeri, S., & Harvey, C. D. (2017). Distinct timescales of population coding across cortex. *Nature*, *548*(7665), 92.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., et al. (2016). Progressive neural networks. arXiv preprint arXiv:1606.04671.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *Proceedings of the 33th international conference on machine learning* (pp. 842–1850).

Schmidhuber, J. (1991). Reinforcement learning in Markovian and non-Markovian environments. In *Advances in neural information processing systems* (pp. 500–506).

Shibata, K., & Sakashita, Y. (2015). Reinforcement learning with internal-dynamics-based exploration using a chaotic neural network. In *Neural networks (IJCNN), 2015 international joint conference on* (pp. 1–8). IEEE.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354.

Silver, D. L., Yang, Q., & Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms.. In *AAAI spring symposium: lifelong machine learning, Vol. 13* (p. 05).

Smith, M. A., Ghazizadeh, A., & Shadmehr, R. (2006). Interacting adaptive processes with different timescales underlie short-term motor learning. *PLoS Biology*, *4*(6), e179.

Softky, W. R., & Koch, C. (1993). The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. *Journal of Neuroscience*, *13*(1), 334–350.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction, Vol. 1*. MIT press Cambridge.

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

Tanaka, S. C., Doya, K., Okada, G., Ueda, K., Okamoto, Y., & Yamawaki, S. (2016). Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. In *Behavioral economics of preferences, choices, and happiness* (pp. 593–616). Springer.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., & Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI conference on artificial intelligence*.

Thrun, S., & Mitchell, T. M. (1994). *Learning one more thing: Technical Report*, Carnegie-Mellon Univ Pittsburgh PA Dept of computer science.

Thrun, S., & Pratt, L. (1998). Learning to learn: Introduction and overview. In *Learning to learn* (pp. 3–17). Springer.

Uhlenbeck, G. E., & Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical Review*, *36*(5), 823.

Utsunomiya, H., & Shibata, K. (2008). Contextual behaviors and internal representations acquired by reinforcement learning with a recurrent neural network in a continuous state and action space task. In *International conference on neural information processing* (pp. 970–978). Springer.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., et al. (2017). Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 3540–3549). JMLR. org.

Vu, E. T., Mazurek, M. E., & Kuo, Y.-C. (1994). Identification of a forebrain motor programming network for the learned song of zebra finches. *Journal of Neuroscience*, *14*(11), 6924–6934.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., et al. (2017). Sample efficient actor-critic with experience replay. In *Proceedings of the international conference on learning representations*.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., et al. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, *21*(6), 860.

Wierstra, D., Foerster, A., Peters, J., & Schmidhuber, J. (2007). Solving deep memory POMDPs with recurrent policy gradients. In *International conference on artificial neural networks* (pp. 697–706). Springer.

Yamashita, Y., & Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLoS Computational Biology*, *4*(11), e1000220.

Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., & Ahn, S. (2018). Bayesian model-agnostic meta-learning. In *Advances in neural information processing systems* (pp. 7332–7342).

Zhang, M., McCarthy, Z., Finn, C., Levine, S., & Abbeel, P. (2016). Learning deep neural network policies with continuous memory states. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 520–527). IEEE.