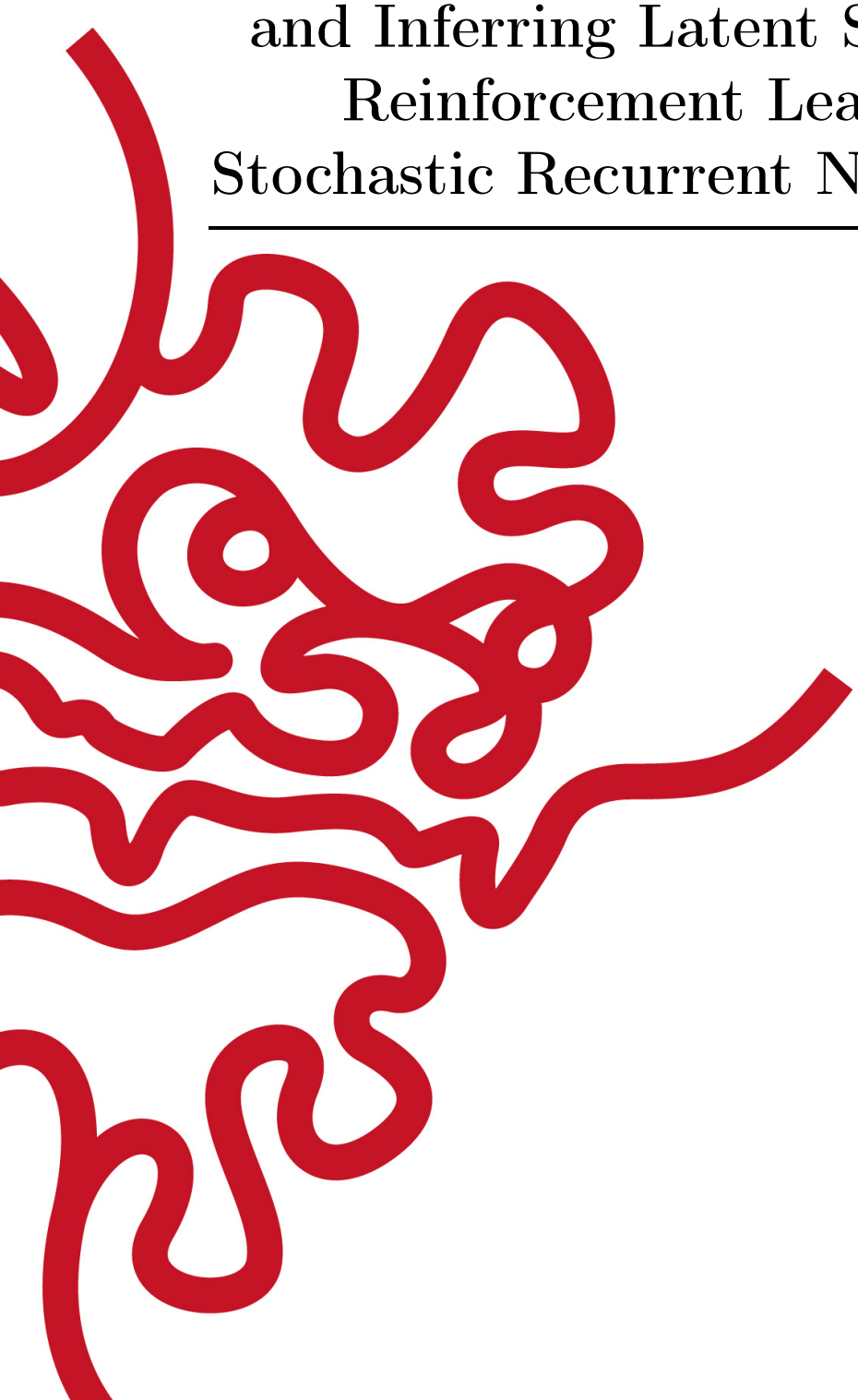
Okinawa Institute of Science and Technology
Graduate University

Thesis submitted for the degree

Doctor of Philosophy

Self-Organization of Action Hierarchy and Inferring Latent States in Deep Reinforcement Learning with Stochastic Recurrent Neural Networks



by

Dongqi Han

Supervisor: **Jun Tani**
Co-Supervisor: **Kenji Doya**

September, 2022

Declaration of Original and Sole Authorship

I, Dongqi Han, declare that this thesis entitled *Self-Organization of Action Hierarchy and Inferring Latent States in Deep Reinforcement Learning with Stochastic Recurrent Neural Networks* and the data presented in it are original and my own work.

I confirm that:

- No part of this work has previously been submitted for a degree at this or any other university.
- References to the work of others have been clearly acknowledged. Quotations from the work of others have been clearly indicated, and attributed to them.
- In cases where others have contributed to part of this work, such contribution has been clearly acknowledged and distinguished from my own work.
- None of this work has been previously published elsewhere, with the exception of the following:
  - Dongqi Han, Kenji Doya, and Jun Tani. "Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks." *Neural Networks* 129 (2020): 149-162.
  - Dongqi Han, Kenji Doya, and Jun Tani. "Variational recurrent models for solving partially observable control tasks." *International Conference on Learning Representations (ICLR)*, 2020.
- The first article above have been published in an open access format under the 'Creative Commons Attribution 4.0 International' license[1]. I hold the copyright of the second article above[2]. I have permission to reprint them for the purpose of the thesis.

Date: September, 2022
Signature: Dongqi Han

[1] https://creativecommons.org/licenses/by-nc-nd/4.0/
[2] https://iclr.cc/FAQ/Copyright

Abstract

The thesis aims to advance cognitive decision-making and motor control using reinforcement learning (RL) with stochastic recurrent neural networks (RNNs). RL is a computational framework to train an agent, such as a robot, to select the actions that maximize immediate or future rewards. Recently, RL has undergone rapid development by introducing artificial neural networks as function approximators. RL using neural networks, also known as deep RL, have shown super-human performance on a wide range of virtual and real-world tasks, such as games, robotic control, and manipulating nuclear fusion devices. There would not be such a success without the efforts of numerous researchers who developed and improved the deep RL algorithms. In particular, most of the works focus on designing or revising the RL objective functions by mathematical analysis and heuristic ideas. While the well-formulated loss functions are critical to the RL performance, relatively fewer efforts have been paid to developing and improving the architecture of the neural network models used in deep RL. The thesis discusses the benefits of using novel network architectures for deep RL. In particular, the thesis includes two of the authors' original studies about developing novel stochastic RNN architectures for RL in partially observable environments. The first work proposes a novel, multiple-level, stochastic RNN model for solving tasks that require hierarchical control. It is shown that an action hierarchy, characterized by consistent representation for abstracted sub-goals in the higher level, self-develops during the learning in several challenging continuous robotic control tasks. The emerged action hierarchy is also observed to enable faster relearning when the sub-goals are recomposed. The second work introduces a variational RNN model for predicting state transitions in continuous robotic control tasks in which the environmental state is partially observable. By predicting subsequent observations, the models learn to represent the underlying states of the environment that are indispensable but not observable. A corresponding algorithm is proposed to facilitate efficient learning in partially observable environments. The proposed studies suggest that the performance of RL agents can be improved by adequate usage of stochastic RNNs structures, which provides novel insights for designing better model architectures for future deep RL studies.

Acknowledgment

I cannot remember how many have happened during my Ph.D. life. First, I would like to thank my family: it would be tough for me to study alone in a foreign country without their support.

When I entered OIST initially, everything was fresh. Before joining my current laboratory, I learned so much from the rotation experiences in 3 different units. I would like to thank professor Kenji Doya, my first rotation supervisor and also my Ph.D. co-supervisor, for bringing me into the fantastic world of neuroscience and neural networks. In Doya unit, I also would like to thank Tadashi Kozuno, who taught me a lot of theoretic knowledge for my thesis study. I would also like to thank professor Erik De Schutter, my second rotation supervisor and my Ph.D. mentor. In Erik's lab, I did my first substantial research on modeling spiking neural networks, through which I learned a lot about conducting scientific research. In particular, I would like to thank Dr. Sungho Hong for advising my research in DeSchutter Unit and teaching me how to present my study. Then, I would like to thank professor Denis Konstantinov, my out-of-field rotation supervisor, for instructing me to study an interesting physics problem.

After one year of rotations, I decided to join the Cognitive Neurorobotics Research Unit (CNRU), led by professor Jun Tani, as my thesis lab. At CNRU, I learned about a lot of interesting studies. I would like first to thank professor Jun Tani, my thesis supervisor. He carefully advised my research during my Ph.D. career. He provided numerous exciting and inspiring thoughts and ideas, which reformed my way of thinking. More importantly, his enthusiasm for research has infected me a lot, making me highly motivated when challenging a new and under-explored research topic.

I also would like to thank my lab mates in CNRU with whom I worked for years: Nadine Wirkuttis, Takazumi Matsumoto, Fabien Benureau, Jeffrey Queißer, Wataru Ohata, Prasanna Vijayaraghavan, Vsevolod Nikulin, Federico Sangati, Hiroki Sawada, Jorge Gallego Perez, Alexander Baranski, Siqing Hou, Hendry Ferreira Chame, Ahmadreza Ahmadi, Jungsik Hwang, Minju Jung, Minkyu Choi, Jinho Chung, and our lab administrator Tomoe Furuya. It is always exciting and inspiring to chat with them. I would also like to thank my friends in OIST for their companionship and help: Ke Wang, Shan Zou, Kunlung Li, Tomoya Noma, Osamu Horiguchi, Masakazu Taira, Joel Perez Urquizo – I cannot list them all.

Finally, I would like to thank the kind and amazing Okinawan people, including other OIST members. Life would not be as happy and peaceful without them.

Abbreviations

5-HT	5-hydroxytryptamine (serotonin)
ANN	artificial neural network
BPTT	back-propagation through time
DOF	degrees of freedom
FNN	feedforward neural network
KLD	Kullback-Leibler divergence
LSTM	long short-term memory
MDP	Markov decision process
MLP	multi-layer perceptron
PC	principal component
PCA	principal component analysis
PDF	probability density function
PO	partially observable
POMDP	partially observable Markov decision process
ReMASTER	Recurrent Multi-timescale Actor-critic with STochastic Experience Replay
RNN	recurrent neural network
RL	reinforcement learning
SAC	soft actor-critic
SEM	standard error of the mean
SLAC	stochastic latent actor-critic
t-SNE	t-distributed stochastic neighbor embedding
VAE	variational auto-encoder
VRNN	variational recurrent neural networks
VRM	variational recurrent model
VSD	variational skill discovery

Contents

Declaration of Original and Sole Authorship iii

Abstract v

Acknowledgment vii

Abbreviations ix

Contents xi

List of Figures xv

List of Tables xvii

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Using artificial neural networks to study decision-making . . . 1
  - 1.1.2 Reinforcement learning . . . 2
  - 1.1.3 Reinforcement learning with recurrent neural networks . . . 3
  - 1.1.4 Hierarchical neural networks . . . 4
  - 1.1.5 Stochasticity . . . 5
- 1.2 Overview . . . 5
  - 1.2.1 Abstracting action primitives . . . 6
  - 1.2.2 Inferring unknown environmental state . . . 6
- 1.3 Chapter Arrangement . . . 7

2 Preliminaries 9
- 2.1 Reinforcement Learning . . . 9
  - 2.1.1 Introduction . . . 9
  - 2.1.2 Markov decision process . . . 9
  - 2.1.3 Partially observable Markov decision process . . . 11
  - 2.1.4 Value function, Bellman equation and value-based RL . . . 11
  - 2.1.5 Experience replay and off-policy RL . . . 12
  - 2.1.6 Neural network as function approximators . . . 13
  - 2.1.7 Learning policy function and actor-critic methods . . . 15

2.1.8 Model-based reinforcement learning . . . 15
2.1.9 Hierarchical reinforcement learning . . . 16
2.1.10 Soft actor-critic . . . 16
2.1.11 Twin delayed deep deterministic policy gradient . . . 17
2.2 Recurrent Neural Networks . . . 18
2.2.1 Simple RNN . . . 18
2.2.2 Continuous-time RNN . . . 19
2.2.3 Multiple-timescale RNN . . . 19
2.2.4 Gated RNN . . . 20
2.2.5 Variational RNN . . . 20

3 Self-Organization of Action Hierarchy 23
3.1 Background . . . 23
3.2 Prior Work on Hierarchical RL . . . 25
3.3 ReMASTER . . . 28
3.3.1 Multiple timescale stochastic RNN . . . 29
3.3.2 Reinforcement learning . . . 29
3.3.3 Experience replay with RNN . . . 30
3.3.4 Motor and neuronal noise . . . 31
3.4 Self-organization of action hierarchy using ReMASTER . . . 32
3.4.1 Task settings . . . 32
3.4.2 Off-policy advantage actor-critic . . . 33
3.4.3 Noise scales . . . 34
3.4.4 Hyperparameters . . . 35
3.4.5 Sequential target-reaching task results . . . 35
3.4.6 Consecutive relearning task results . . . 38
3.4.7 Learning new tasks with low-level weights frozen . . . 40
3.4.8 Consistency in representing sub-goals . . . 42
3.4.9 Manipulating agent behaviors by clamping high-level neural states 42
3.4.10 Timescales and discountings . . . 44
3.4.11 Effect of hyperparameters . . . 44
3.4.12 Comparing learning from scratch and relearning . . . 44
3.4.13 Neuronal noise . . . 46
3.4.14 Development of internal representations . . . 46
3.5 Scaling up to more challenging tasks . . . 47
3.5.1 Tasks . . . 48
3.5.2 ReMASTER implementation for the additional tasks . . . 51
3.5.3 Experimental results . . . 52
3.6 Summary . . . 56
3.7 Neuroscience Insights . . . 58
3.7.1 Multiple timescales . . . 58
3.7.2 Discount factor . . . 58
3.7.3 Neuronal noise . . . 59

4 Variational RNN for RL in Partially Observable Environments 61
4.1 Background . . . 61
4.2 Related work . . . 63
4.2.1 Deep RL for POMDP . . . 63
4.2.2 Model-based RL . . . 64
4.2.3 Variational Bayes in RL . . . 64
4.2.4 Probabilistic models for encoding belief states in POMDPs . . . 65
4.3 Methods . . . 65
4.3.1 Variational recurrent state-transition models . . . 65
4.3.2 Reinforcement learning controllers . . . 67
4.3.3 Update-to-data ratio . . . 68
4.3.4 Implementation details . . . 68
4.3.5 Hyperparameters . . . 69
4.4 Environments . . . 70
4.5 Results . . . 70
4.5.1 Alternative algorithms . . . 71
4.5.2 Partially observable classic control tasks . . . 72
4.5.3 Partially observable robotic control tasks . . . 73
4.5.4 Long-term memorization tasks . . . 74
4.5.5 Convergence of the keep-learning VRM . . . 75
4.5.6 Ablation study . . . 75
4.5.7 Visualization of trained agents . . . 77
4.5.8 Model accuracy . . . 77
4.5.9 Sensitivity to hyperparameters of the VRMs . . . 77
4.6 Summary . . . 78
4.7 Discussion . . . 80
4.7.1 Model-based and model-free RL . . . 80
4.7.2 Representation learning and RL . . . 81

5 Conclusion and Future Work 83
5.1 Conclusion . . . 83
5.2 Future work . . . 84

Bibliography 87

List of Figures

2.1 Elements of reinforcement learning 10

3.1 The basic structure of MTSRNN 28
3.2 Task settings 32
3.3 The sequential target-reaching task 35
3.4 Analysis of the sequential target-reaching task using ReMASTER . . . 37
3.5 The consecutive relearning task 39
3.6 Analysis of ReMASTER agents 41
3.7 Manipulating agent behaviors by clamping high-level RNN states . . . 43
3.8 Timescale dependence 45
3.9 Sensitivity analysis for hyperparameters 46
3.10 Performance comparison among relearning phases and the control case 47
3.11 Noise dependence of ReMASTER 48
3.12 Neuronal noise ablation study 49
3.13 Development of internal representation for sub-goals 50
3.14 Rendering of the environments 50
3.15 Results of the vision-based sequential reaching task 53
3.16 Results of the robot arm sequential touching tasks 54

4.1 Diagrams of the proposed algorithm 65
4.2 Computation diagram 66
4.3 Learning curves of the classic control tasks 73
4.4 Learning curves of the robotic control tasks 74
4.5 Learning curves of the sequential target reaching task 75
4.6 Relationship between the average return of the agent and loss function 75
4.7 Ablation study 76
4.8 Visualizing behavior of the trained agents 77
4.9 Examples of observation predictions 78
4.10 Sensitivity to hyperparameters for VRM 79

List of Tables

3.1 Hyperparameters for ReMASTER 36
3.2 Consistency of RNN outputs in representing sub-goals 42
3.3 Details of the CNN layers 52
3.4 Hyperparameters of the updated implementations of ReMASTER. . . 55
3.5 Consistency of RNN outputs in representing sub-goals for the additional tasks 56

4.1 Shared hyperparameters 69
4.2 VRM hyperparameters 70
4.3 Information of the environments we used. 70

Chapter 1

Introduction

1.1 Motivation

1.1.1 Using artificial neural networks to study decision-making

Learning to perform adaptive and intelligent decision-making and motor control is a central topic in understanding intelligence. Scientists have tried to approach the problem from various points of view.

Neuroscientists study this by conducting experiments on animal and human subjects. Since the central nervous system plays a significant role in decision-making, researchers have been studying the brain for more than a hundred years [56]. The scale of study includes molecules, ion channels, cells, layers, neural circuits, and the whole brain. The development of experimental methodologies has allowed researchers to obtain a deep understanding of the mechanisms at the microscopic scale of neurons using, e.g., electrodes or calcium imaging, or at the macroscopic scale of brain regions using, e.g., functional magnetic resonance imaging [77]). While the experimental neuroscientific studies are undoubtedly fundamental and vital, there are practical challenges in-between cell-level mechanism and macroscopic-level imaging: the experimental study of neural circuits faces some major difficulties. First, to study a neural circuit, we need to know the activity of each neuron at the microscopic level. Since these neurons are usually distributed not on the same plane or in a small cube, existing techniques such as calcium imaging could not cover all the neurons of the circuit without causing considerable damage to the brain tissue. Second, the brain regions are connected with each other. It is non-trivial to consider the impact of the interconnectivity between the region of interest and the other neural circuits. Moreover, even if we could record the activities of all neurons and synapses in a network, the size of the data stream could be immense, given that the activities changes at the millisecond level. This yields a considerable challenge to data saving and processing. Therefore, there has been a long-standing gap between macro-level animal behavioral study and microscopic neuroscience experimental study.

Facing such difficulties, computational neuroscience [188] tries to investigate the neural circuits with mathematical models based on the fundamental neural mechanisms found in experiments. A neuron is usually modeled as an electric circuit characterized by its membrane potential. By computer simulation, researchers can access or record

the status of any neuron or synapse at any time. Thus it is convenient to analyze the dynamical process in the networks. Computational neuroscience has been successfully applied to many important aspects of decision-making and motor control, such as spikes propagation [43, 171], working memory [36], etc. However, when it comes to learning, computational models often face difficulties: although we know some basic principles of learning in biological brains, such as spike-time-dependent plasticity [140] and activity-dependent plasticity [75], empirical results demonstrate considerable practical difficulties in training a network model to tackle challenging tasks by biologically-plausible learning rules [212]. So far, studying sophisticated behavior learning with biologically-plausible neural networks seems unpractical.

On the other hand, the recent development of artificial intelligence has demonstrated the advantage of backpropagation [112, 176] as the core learning algorithm for artificial neural networks (ANN). Learning with ANNs and backpropagation, or *deep learning* [184], has proven successful in various challenging tasks, including but not limited to game playing [196, 234], protein structure prediction [106], and natural language processing [249]. Moreover, recent studies by Lilicrap and collegues [135, 136], Song et. al. [203] and Whittington & Bogacz [243] argue that even though learning in the brain does not work in exactly the same way as backpropagation in ANNs, the brain is still able to implement the core principles underlying backpropagation, and biological learning may locally approximate the backpropagation algorithm with feedback connections. Thus, while neuroscience studies have built up the foundation of understanding human-like intelligence, studying decision intelligence in ANNs appears as a complementary approach to deepen our understanding of the intelligence of animals and humans because of its learning efficiency in relatively complicated tasks. This thesis will focus on learning to perform cognitive decision-making and motor control using ANN models.

1.1.2 Reinforcement learning

Armed with ANN as a powerful tool, an essential problem is the learning principle(s) for acquiring decision-making and motor control skills. Reinforcement learning (RL) is one of the most established frameworks for action learning [208]. In RL, the agent tries to obtain more (long-term) rewards by improving its strategy (or *policy* in RL terminology). RL has undergone rapid development in recent years [208]. In particular, deep RL, i.e., RL using ANNs, has proven successful on highly challenging decision-making tasks [197, 234]. The fundamental algorithm in RL, temporal difference learning [211], has also been used to model the reward-relevant learning process in the brain [158]. In the animal brain, the neurotransmitter *dopamine* [245] represents reward prediction error signals [146] that affect decision-making and learning [59].

Learning needs experience. In RL, the experience for learning comes from the interaction between an agent and the environment by performing a combination of random actions and a learned control strategy. The agents need to trade-off between exploring the environment for novel experiences and exploiting learned knowledge to obtain more returns. Such an exploratory scheme is an essential basis for learning in humans and animals.

1.1.3 Reinforcement learning with recurrent neural networks

This thesis tries to advance intelligent decision-making and motor control by modeling robots[1] using RL with ANNs. In particular, recurrent neural networks (RNN) models [50] are used for this purpose.

Most deep RL studies used feedforward neural networks (FNN) as function approximators of value, policy and etc., with a continuous-valued state as input. In an MDP, an FNN should, if it is expressive enough, be able to act as a state-action/state value function. This is because only the current state (observation) and policy are needed to evaluate expected reward and return, and thus it is straightforward to use an FNN such as a convolutional neural network (CNN) in RL [131, 149].

However, in more general real-world applications, we often need to consider partially observable environments (where the agent cannot directly observe the full underlying state that determines the system dynamics) [107]. A category of partially observable tasks common in real life is history-dependent, i.e., state transition and reward function also depend on previous observations and actions. Therefore, the state-transition history should be taken into account. Previous studies on history-dependent tasks have covered real-time vision-based robotic control [86, 125] and video games that need memorization of previous events [103, 110, 234].

In deep RL, a straightforward solution to history-dependent tasks is to include previous observations in the input to value/policy function approximators. To overcome the problems of over-extensive history or history of variable length, RNNs are naturally considered a replacement for FNNs. At each step of interacting with the environment, an RNN takes current observation as input and extracts the observed information into its hidden state. If the RNN is well trained, its hidden state should carry critical information that underlies state transition and reward function. Then the hidden state can be used for computing any quantity that depends on historical observations.

Moreover, RNNs are particularly useful for continuous-time sensorimotor tasks by discretizing the time to discrete steps, as they can deal with an arbitrary sequence length in principle[2].

Early attempts to use RNNs in RL can be traced back to the 1990s [181, 183, 228, 244], which employed simple RNNs and investigated only simple tasks because of limited computation power at that time, as well as the vanishing gradient problem [96, 97]. After modern RNN architectures such as long short-term memory (LSTM) [98] and continuous-time RNN [22] were developed, together with the development of computation infrastructure and other deep learning techniques [93, 119], RNNs have been applied to much more challenging RL tasks, such as playing games with hidden information and high-dimensional action and state space [131, 234] and have achieved super-human performance.

In most of these works, RNNs only play the role of hidden representation learning

[1] In this thesis, a *robot* is defined as an autonomous agent with sensors to perceive the information of the environment and with the capacity to conduct computations to perform motor control so as to interact with the environment.

[2] Although the neural ordinary differential equation [27] is a more specified model for continuous-time environmental dynamics, it does not fit the (partially observable) Markov decision process where the time steps are discrete.

and function approximators to deal with memory-dependent tasks [92, 94, 110, 234, 255]. However, empirical work shows that even for the environments which usually were considered fully observable, such as Atari games [149], extraordinary performance can be achieved by employing RNNs [110]. It remains under-explored how RNNs facilitate RL.

It was also proposed that in RL, RNNs enable meta-learning (i.e., learning a range of tasks that share some common properties) [6, 238]. Wang et al. [238] argued that the prefrontal cortex, which has many recurrent connections, plays a key role in meta-learning. The authors demonstrated that an RNN network can perform a number of RL tasks even without synaptic learning by updating task-relevant latent variables in hidden units. Also, Al-Shedivat et al. [6] demonstrated that robotic agents using RL with RNNs can perform meta-learning in dynamically changing tasks.

An important feature of RNNs is that it maintains an internal state, which makes an RNN beyond an input-output machine. Although the hidden state of feedforward networks (values in the hidden layers) can also be meaningful or interpretable [119, 170], they are only able to represent information from the current input instead of the contextual information. However, a trained RNN's internal state is capable of encoding information from both current and historic inputs (and future input if using bi-directional RNNs). Thus taking advantage of the internal state has also led to an interesting direction in recent years. For example, Lee et al. [125] trained a recurrent state-transition model and used its internal state as additional input to the value network, which enabled the value function to capture contextual information.

1.1.4 Hierarchical neural networks

We may get inspiration from intelligent animals, such as mammals, for developing an intelligent AI for decision-making or motor control. In particular, mammals' behaviors usually appear as the composition of reusable movement skills, e.g., jumping, running, and pushing [179, 222]. Each movement skill includes detailed actions of many parts of the body. Once a movement skill is acquired, the animal may use it in unseen tasks with little requirement of re-adapting the detailed actions.

Understanding the brain mechanisms behind the flexibility and compositionality of mammals' actions should be beneficial to creating an AI model with a similar ability. In the brains of mammals, many neural pathways consist of multiple brain regions with hierarchical intrinsic neural properties [20, 54, 95, 154]. The lower-level brain regions deal more with reusable primitives or features, and the higher-level ones with the composition or integration of them. For example, in the visual pathway, the primary visual cortex (V1), which receives vision signals earlier, has a smaller receptive field compared to the deeper regions such as visual area V4 [213]. For action generation, there is also neuroscientific evidence that the motor pathway consists of brain areas focusing on more abstracted-level and detailed-level actions, respectively. For example, Shima & Tanji [192, 193, 218] suggested that the supplementary motor area deals with abstract movement sequence processing and the primary motor cortex with detailed movement patterns.

For ANN models dealing with vision, one of the most popular architectures, the CNN, is characterized by spatial hierarchy utilizing intrinsic spatial scale constraints

assigned among multiple layers [119, 170]. However, relatively few studies worked on developing brain-inspired hierarchical neural network architecture for decision-making/motor control tasks. A classic work by Yamashita & Tani [248] developed a multi-layer RNN known as *multiple-timescale RNN* (MTRNN, Chap. 2.2.3) with a hierarchy of intrinsic timescales, inspired by neuroscientific and behavioral studies that suggested more abstracted actions correspond to slower timescales [101, 154, 156, 199]. It was shown that a hierarchical representation in the network emerged via supervised learning, where contiguous sequences of actions are split into reusable primitives, which in turn are flexibly integrated into new sequences. However, Yamashita & Tani's work [248] required human demonstrations. It has not been investigated how MTRNN contributes to the autonomous development of motor skills in self-exploratory learning with environmental rewards (RL). This is one of the major motivations for our work proposed in Chap. 3.

1.1.5 Stochasticity

The randomness in neural activities is also considered an important character in cognition. In the brain, it is known that cortical neurons, which play a key role in cognition, have stochastic firing properties, such as irregular inter-spike intervals and noisy firing rates [14, 15, 91, 201]. For machine learning, Bayesian models such as the variational auto-encoder [115] have been shown to effectively extract the probabilistic structure hidden in spatial patterns of the data.

Dynamically changing environments in the real world are often stochastic in terms of state transition and/or observation. Variational RNNs [34, 78] are powerful architectures to model stochastic dynamics. By modeling the sequences of, e.g., environmental observations, variational RNN models can learn to better represent the probabilistic structure underlying the environment dynamics, which also provides convenience and flexibility for RL [88, 125]. The application of variational RNN in RL will be addressed in Chap. 4.

Moreover, while most RL methods apply noise to actions to achieve random exploration, it is also possible to consider a stochastic model [61, 87] that generates up-stream noise for exploration. If the model captures a high-level symbolic representation of abstracted actions, random noise in the model may facilitate exploration in the abstracted action space: an example will be addressed in Chap. 3.

1.2 Overview

The proposed thesis study aims to investigate the role(s) of RL with stochastic RNN in acquiring cognitive decision making and motor control functions in artificial agents. Two studies using RL with stochastic RNN on robotic tasks in partially observable environments are addressed. These topics involve essential aspects of learning to perform effective and adaptive decision-making and motor control: abstracting action primitives and inferring the unknown environmental state. In addition, the connections between the thesis studies and neuroscience/cognitive science will also be discussed.

1.2.1 Abstracting action primitives

In this study, a novel framework of RL using a multiple-timescale, stochastic RNN model is proposed. The framework is characterized by two essential ideas. The first is to have an intrinsic hierarchy of timescales in the network, where the lower level neural activities are subject to faster dynamics than those of the higher level, inspired by findings in neuroscience [25, 101, 154, 156, 177, 199]. The second idea is to introduce stochasticity in the neuronal activities, reflecting the fact that neurons in the cortex are featured with highly stochastic activities [14, 15, 91, 201]. Experiments are conducted in several challenging continuous control tasks with an interpretable task hierarchy. The results show that the RNN model autonomously learns to abstract sub-goals and thus self-develops the internal representation of an action hierarchy in the internal dynamics. Furthermore, it is shown that the self-developed compositionality of the network facilitates more efficient relearning when the agent adapts to a novel task that is a re-composition of previously learned sub-goals than learning from scratch. A performance gain is also observed when neural activities are subject to stochastic rather than deterministic dynamics.

This work has been published as:

- Dongqi Han, Kenji Doya, and Jun Tani. "Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks." *Neural Networks*, 129:149–162, 2020.

1.2.2 Inferring unknown environmental state

This study attempts to demonstrate more efficient and robust learning in partially observable environments, in which deep RL agents usually perform poorly. One significant difficulty originates from that two problems need to be handled simultaneously: how to extract information from the original observation to solve the task and how to improve the acting strategy (policy). In this work, an RL methodology for solving partially-observable tasks is proposed, which consists of two parts: a variational RNN for modeling the environment and an RL controller that takes the inputs from the environment and the RNN model. The proposed method is tested in two types of partially observable robot control tasks, in which the position or velocity components of the state variables are not observable, and a task that requires long-term memory. In challenging tasks wherein the underlying environmental states cannot be inferred from raw observations in a straightforward manner, the proposed method demonstrated more efficient learning than alternative approaches.

This work has been published as:

- Dongqi Han, Kenji Doya, and Jun Tani. "Variational recurrent models for solving partially observable control tasks." *International Conference on Learning Representations (ICLR)*, 2020.

1.3 Chapter Arrangement

The remaining chapters of this thesis are arranged as follows. Chap. 2 introduces essential preliminary knowledge of the thesis. Then, Chap. 3, 4 introduce the aforementioned two studies, respectively. The last chapter concludes the thesis and discusses possible future directions.

Chapter 2

Preliminaries

2.1 Reinforcement Learning

2.1.1 Introduction

Reinforcement learning (RL) is a machine learning scheme in which agents learn the best policy to maximize long-term rewards in a given environment [208]. In RL, the agents explore the environment and generate data by receiving rewards or penalties (negative rewards) from interaction with the environment.

As figure(2.1) shows, a typical RL framework is comprised of the *agent(s)* and the *environment*, where the agent executes actions in an environment according to its *policy*. By interaction with the environment, the agent observes environmental information, such as position, image, and sound. This information is called *state* in RL. In addition, a reward (or punishment) signal is provided by the environment. A positive reward will be given if the agent finishes a task, performs well, or wins a game. Also, a negative reward is presented when the agent executes poorly or fails to complete the task.

The goal of RL is to learn the optimal policy that leads to the largest benefits. Here *benefits* can be defined as the mean reward, the accumulated rewards, or other measurements of total rewards, depending on the learning task. An RL problem is usually formulated as a Markov decision process (MDP) or a partially observable decision process (POMDP).

2.1.2 Markov decision process

A *Markov decision process* (MDP) [16, 99] is a mathematical framework to describe the process in which a decision maker (the agent in RL) and an environment interact. Outcomes are partly decided by the decision maker's action and are partly stochastic. MDPs are built on discrete, consecutive time steps: $t = 0, 1, 2, \ldots$.

Mathematically, an MDP is defined as a 5-tuple $(\mathbb{S}, \mathbb{A}, P, R, \gamma)$:

- $\mathbb{S}$ is a set of states, called *state space*.
- $\mathbb{A}$ is a set of actions, called *action space*.

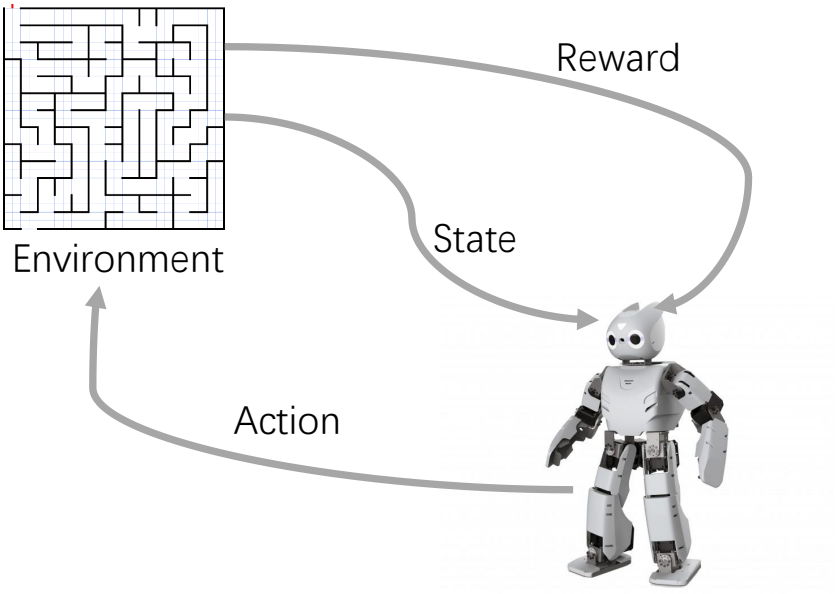


Figure 2.1: Elements of reinforcement learning

- $P(s', s, a) = \Pr[s_{t+1} = s' | s_t = s, a_t = a]$ is a mapping from $\mathbb{S} \times \mathbb{S} \times \mathbb{A}$ to $[0, 1]$, which indicates the probability that action a at time t when $s_t = s$ will result in $s_{t+1} = s'$. It is called the *transition probability*.
- $R(s', s, a)$ is a mapping from $\mathbb{S} \times \mathbb{S} \times \mathbb{A}$ to $\mathbb{R}$, where $\mathbb{R}$ denotes real numbers. $R(s', s, a)$ indicates the immediate reward (or expected immediate reward) received when action a leads to the state transition s to s'. R is called the *reward function*. The reward at step t is denoted by r_t.
- $\gamma \in [0, 1]$ is called the *discount factor*, which decides the trade-off between immediate reward and future reward.

The action acts according to a strategy function $\pi : \mathbb{A} \times \mathbb{S} \to [0, 1]$, called *policy*, which is the probability of executing an action a when the current state is s. The goal of an MDP is to develop a policy π that maximizes the expectation (denoted by $\mathbb{E}$) of the sum of discounted rewards when the agent executes actions following policy π:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{2.1}$$

$$= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, s_t, a_t) \right], \tag{2.2}$$

where the action at each step is sampled from the policy: $a_t \sim \pi(a_t | s_t)$. The expectation is taken over $s_{t+1} \sim P(s_{t+1}, s_t, a_t)$. Eq. 2.1 provides an intuitive understanding of the discount factor γ — an MDP with a larger γ considers longer-term rewards. Note that here we have omitted the dependence on the initial state s_0 for simplicity. If s_0 follows some distribution, the expectation should also conditioned on this distribution.

The policy π^* that maximizes equation 2.1 is called the *optimal policy*.

It is called *model-based* RL if the transition probability $P(s', s, a)$ is known or estimated by a world model of the agent. Otherwise, it is referred to as *model-free* RL.

2.1.3 Partially observable Markov decision process

More general cases of MDPs can be defined as *partially observable Markov decision processes* (POMDPs) [107], in which some part of the environmental state is not observed by the agent. The unobservable *underlying* state may be important in the state-transition dynamics of the environment. MDP can be considered a special case of POMDP.

A 7-tuple $(\mathbb{S}, \mathbb{A}, P, R, \mathbb{X}, O, \gamma)$ can be used to describe a POMDP. $\mathbb{S}, \mathbb{A}, P, R, \gamma$ are defined in the same way as in an MDP. However, the agent cannot directly obtain a state s but an observation x from the environment. The observation in a POMDP usually contains partial information of the environment state. The set of all possible observations is denoted by $\mathbb{X}$. The probability that the agent observes an observation is given by the observation probability function $O : \mathbb{S} \times \mathbb{A} \to p(\mathbb{X})$. The objective of a POMDP is similar to that of an MDP: to maximize Eq. 2.1.

2.1.4 Value function, Bellman equation and value-based RL

Learning to estimate Eq. 2.1 is the basis of many RL algorithms, which are built on MDPs and POMDPs. For this purpose, the *state value function* $V_\pi(s)$ is introduced as the expected sum of discounted rewards under policy π from the current state s:

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]. \tag{2.3}$$

Note that the state value function is conditioned on policy $\pi(a|s)$, therefore, learning to estimate V_π is known as *policy evaluation*. Note that we have

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[\sum_{\tau=0}^{\infty} \gamma^\tau r_{t+\tau} | s_t = s] && (2.4)\\ &= \mathbb{E}_\pi[r_t + \gamma \sum_{\tau=0}^{\infty} \gamma^\tau r_{\tau+1} | s_t = s] && (2.5)\\ &= \mathbb{E}_\pi[R(s_{t+1}, s_t, a_t) + \gamma V_\pi(s_{t+1}) | s_t = s] && (2.6)\\ &= \sum_a \pi(a|s) \sum_{s'} \left[P(s', s, a) R(s', s, a) + \gamma V_\pi(s')\right]. && (2.7) \end{aligned}$$

Eq. 2.7 is called the *Bellman equation for the state value function* [18, 208]. Many policy evaluation algorithms are based on the Bellman equation or its variants. The readers may gain some intuitive understanding of how policy evaluation works from Algorithm 1, which is a simple example of iteratively updating the tabular value function for discrete action and state spaces under a given policy π [208].

Algorithm 1 Iterative policy evaluation for discrete action and state spaces

Input: policy $\pi(a|s)$, state-transition function $P(s', s, a)$, reward function $R(s', s, a)$
Initialization a table $V(s) = 0$ for all $s \in \mathbb{S}$
while $V(s)$ does not converge **do**
  for each $s \in \mathbb{S}$ **do**
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P(s', s, a)[R(s', s, a) + \gamma V(s')]$
  end for
end while

With a value function, an essential problem is how to choose good actions. One common approach to address this question is to introduce an *action-value function*, or *Q-function* under policy π, defined as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a] \tag{2.8}$$

$$= \mathbb{E}_\pi[R(s', s, a) + \gamma \sum_{\tau=0}^{\infty} \gamma^{\tau+1} r_{\tau+1} | s_0 = s] \tag{2.9}$$

$$= \sum_{s'} P(s', s, a)[R(s', s, a) + \gamma V_\pi(s')]. \tag{2.10}$$

A Q-function provides a measurement of expected future returns with regard to a state-action pair (s, a). For a task with a discrete action space, it is straightforward to obtain the "greedy" action at state s with a Q-function:

$$a_{greedy}(s) = \text{argmax}_{a'} Q_\pi(s, a'). \tag{2.11}$$

2.1.5 Experience replay and off-policy RL

Like humans, RL agents can also utilize past experiences to improve their current policy, namely *experience replay*. If the experience used for training is collected following a different policy from the current policy (or the policy one wants to optimize), it is called *off-policy* RL; otherwise, *on-policy* RL. Usually, an RL agent keeps updating its policy from time to time, so experience replay normally needs to use off-policy RL algorithms. There are both pros and cons of experience replay with off-policy data. The goodness is that there are more data to train the model since the old data can be reused. Moreover, the data often cover a wider part of the state space than only on-policy data. This makes it less possible for an agent to be trapped around a local optimum, thanks to the experiences of farther positions in the state space. The badness is that it is computationally less straightforward to optimize the value and policy functions. The following will introduce why.

Let us first consider learning a state value function $V_\pi(s)$ with on-policy RL. Suppose we have a buffer $\mathcal{B}$ to store the agent's experiences for learning. In on-policy cases, the experiences in $\mathcal{B}$ are collected using the current policy π. Then, estimating V_π will be

straightforward using Algorithm 1:

$$V_\pi(s_t) \leftarrow \mathbb{E}_\pi[R(s', s_t, a_t) + \gamma V_\pi(s')] \tag{2.12}$$

$$\approx \mathbb{E}_{(s_{t+1}, s_t, r_t)\sim\mathcal{B}}[r_t + \gamma V_\pi(s_{t+1})], \tag{2.13}$$

where $(s_{t+1}, s_t, r_t) \sim \mathcal{B}$ means the state-transition tuple (s_{t+1}, s_t, r_t) is randomly sampled from the buffer $\mathcal{B}$ with equal probabilities. Note that this approximation (Eq. 2.13) only works with on-policy RL. However, suppose the experience in $\mathcal{B}$ was generated using a different policy from the target policy π, namely in the off-policy case. In that case, we cannot simply sample data from the buffer to update the value function (like in Eq. 2.13) since the expectations are not based on the same distribution.

Fortunately, this problem can be overcome using algorithms designed for off-policy RL. One of the most popular algorithms is known as *Q-learning* [241]. In many RL problems, we only need the optimal policy π^*. Q-learning directly estimates the Q-function of the optimal policy π^* using the experiences in the replay buffer $\mathcal{B}$:

$$Q^*(s_t, a_t) \leftarrow Q^*(s_t, a_t) + \alpha\left(r_t + \gamma \text{max}_{a'} Q^*(s_{t+1}, a') - Q^*(s_t, a_t)\right), \quad (s_{t+1}, s_t, a_t, r_t) \sim \mathcal{B}, \tag{2.14}$$

where $0 < \alpha <= 1$ is the learning rate. The term $r_t + \gamma \text{max}_{a'} Q^*(s_{t+1}, a') - Q^*(s_t, a_t)$ is called *temporal difference (TD-error) in Q-learning*, which reflects the approximation error of the Q-function. However, in general cases, TD-error may refer to the approximation error of either the state value function or Q-function based on a Bellman equation.

In many practical scenarios, the agent alternately interacts with the environment to get experiences and updates its policy. This is known as *online RL*. The opposite of online RL is *offline RL*, where the agent cannot interact with the environment, and the policy needs to be learned with a given dataset of experiences. In online RL, the agent needs some degrees of exploratory, random actions to avoid being trapped in a local optimum. Using *ϵ-greedy* policy of a Q-function is a common practice for Q-learning:

$$a = \begin{cases} \text{argmax}_{a'} Q(s, a') & \text{if } \eta \sim \text{Uniform}[0, 1] > \epsilon, \\ \text{a random action in } \mathbb{A} & \text{otherwise,} \end{cases} \tag{2.15}$$

where η is a random variable sampled every time when an action is chosen, and ϵ is usually a small, positive number (such as 0.1). A practical Q-learning procedure for online RL in MDPs with discrete state and action spaces is described in Algorithm 2. Q-learning can be straightforwardly generalized to MDPs with continuous state spaces [149] using K function approximators $Q_k(s)$, $k = 1, 2, ..., K$ (e.g., linear functions), where each $Q_k(s)$ corresponds to an action $a \in \mathbb{A}$, and K is the size of the action space.

Note that we do not need to explicitly learn a policy function $\pi(a|s)$ in Q-learning, since it can be directly inferred using Eq. 2.11 or 2.15. The RL methods that only require learning value functions are referred to as *value-based* methods.

2.1.6 Neural network as function approximators

Early RL methods that worked well were mostly designed for discrete state and action space using tabular value function and/or policy function. However, the state space

Algorithm 2 Q-learning for discrete state and action spaces

Input: an MDP environment with the initial state s_0 and a learning rate α
Initialize a Q-table $Q^*(s,a) = 0$ and an empty replay buffer $\mathcal{B}$.
while task not mastered **do**
  (Interacting)
  Sample an action based on current state: $a_t \sim \epsilon - greedy(Q(s_t, a_t))$ (Eq. 2.15)
  Execute action a_t for one environment step and observe r_t and s_{t+1}.
  Record the state-transition tuple (s_t, s_{t+1}, a_t, r_t) into $\mathcal{B}$
  $t \leftarrow t + 1$
  (Learning)
  Randomly sample a batch of tuples $(s_{\tau_i}, s_{\tau_i+1}, a_{\tau_i}, r_{\tau_i})$, $i = 1, 2, ..., N$ from $\mathcal{B}$
  for $i = 1, 2, ...N$ **do**
  $Q^*(s_{\tau_i}, a_{\tau_i}) \leftarrow (1-\alpha)Q^*(s_{\tau_i}, a_{\tau_i}) + \alpha\left(r_{\tau_i} + \gamma \text{max}_{a'} Q^*(s_{\tau_i+1}, a')\right)$
  end for
end while

is often continuous in many problems. Traditionally, linear approximators were often used to model the value and policy functions for a continuous state space [13]. For example, $V(\boldsymbol{s}) = \boldsymbol{\phi} \cdot \boldsymbol{s} + b$, where $\boldsymbol{s}$ is the state vector, and $\boldsymbol{\phi}$ and b are learnable parameters.

Linear approximators require fewer computational resources and can be solved by analytic methods, such as linear regression. However, in complicated tasks, the linear representation of the value function generally cannot give an adequate approximation of the true value function due to its limited *expressive power* (i.e., how well a parameterized function can approximate the true function).

In recent years, RL using artificial neural network (ANN) models as function approximators, a.k.a. *deep RL*, has undergone rapid development [148, 149, 240], thanks to the expressive power of ANN and the corresponding training techniques such as stochastic gradient descent [21]. One famous example is AlphaGo [196, 197], which took advantage of deep convolutional neural networks to achieve super-human performance in the game of *Go*.

Neural networks can be roughly divided into two classes: *feedforward neural networks* (FNNs) and *recurrent neural networks* (RNNs). FNNs are networks that do not have cycles or loops in the network [254], processing information in a single direction (from input to output). Convolutional neural networks [124] and multi-layer perceptrons [120] are examples of FNNs. The other class, RNNs [80], have self-feedback connections between neurons. This feature enables RNNs to use their hidden states[1] to process sequential inputs. Popular RNNs include *Elman-type* RNN [50], *long short-term memory* (LSTM) [98], *gated recurrent unit* (GRU) [30], etc. A well-trained RNN can encode inputs from previous time steps (orders in the sequence) as memory-like hidden states in the network, such as the work by Utsunomiya & Shibata [228] (See Chap. 2.2 for a more detailed introduction of RNN).

Many deep RL methods are built on MDPs, where an FNN can be a useful function

[1]The hidden states of an RNN means values of neuronal activities in hidden layers of the RNN. These should not be confused with the state s in an MDP/POMDP.

approximator for value functions and policy functions using current state s_t as input since the state transition function and reward function only depend on the current state. However, in a more general situation (e.g., POMDP), where historical observations also need to be taken into account, RNNs are often more suitable for the function approximators.

2.1.7 Learning policy function and actor-critic methods

Value-based methods make it possible to solve some RL tasks without the necessity of explicitly learning the policy function $\pi(a|s)$. However, they also have limitations. One major problem is that when the action space is continuous, it is difficult to conduct the max and argmax operations in Q-learning.

An alternative approach is to learn a policy function that is not directly derived from value functions. The policy function can be parameterized by a function approximator $\pi(a|s) = \pi_\theta(a|s)$, where θ denotes the parameters for the policy function approximator, which is usually an ANN. To learn the policy function, an algorithm called *policy gradient* was introduced by Sutton et al. [209]. The aim is to maximize long-term rewards, namely the objective function.

$$J_\pi(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{2.16}$$

by updating the policy π_θ.

The policy gradient algorithm maximizes $J_\pi(\theta)$ by gradient ascent, where the gradient of $J_\pi(\theta)$, or policy gradient [209], can be obtained by:

$$\nabla_\theta J_\pi(\theta) = \mathbb{E}_\pi\left[Q_\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)\right] \tag{2.17}$$

$$\approx \mathbb{E}_{(s_t,a_t)\sim\mathcal{B}_\pi}\left[Q_\pi(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right], \tag{2.18}$$

where $\mathcal{B}_\pi$ is the replay buffer in which the experiences are collected with policy π.

In practice, the value function(s) usually need to be learned in parallel with learning the policy function. This is called *actor-critic* methods in RL. Actor-critic methods have achieved human-level or super-human behavior on different tasks such as Atari games and robotic control [148, 240].

Note that Eq. 2.18 only applies to on-policy RL. However, there are also off-policy actor-critic methods that utilize experience replay [41, 70, 83]. The RL algorithms used in the thesis are all off-policy actor-critic methods, which will be detailed in Chap. 2.1.10, 2.1.11, and 3.4.2.

2.1.8 Model-based reinforcement learning

The recent rapid progress of deep learning research, empowered by both software and hardware innovations, has enabled RL to solve more challenging and practical tasks [197, 234]. While the state-of-the-art ANNs can be used for approximating value and policy functions, the expressive power of ANNs has further stimulated the development of *model-based RL* [207]. A model predicting the state transition (e.g., predicting

subsequent state/observation from current state and action) of the environment can be trained to facilitate RL. This model is also known as the *world model*. The main categories of model-based RL can be summarized as follows.

Dyna-style (dreaming): The learned world model is utilized to generate simulated experiences that can be used by the RL agent to improve its policy in a model-free manner [109, 207]. By doing so, actual interaction with the real environment can be reduced, which is preferable if it is relatively expensive to explore the real environment.

Policy search (planning): This type of model-based RL assumes that the reward function is differentiable and known, or it is learned by the model [42, 73, 164]. Thus gradient ascent to maximize total return can be straightforwardly conducted in the network by treating the actions as the variables to optimize.

Lookahead: The world model can also be used to perform a lookahead search for future outcomes using a given policy, such as Monte-Carlo tree search [185, 196, 197]. The future outcomes can help the agent to better estimate the value of the current state.

These model-based RL methodologies all leverage the prediction functionality (output) of the learned or provided model. However, another possible advantage is that the underlying state of the environment can be better inferred in terms of the internal representation of a state-transition model rather than the raw observation. Chap. 4 will propose a novel RL methodology according to this idea, which learns a model but does not use it for prediction.

2.1.9 Hierarchical reinforcement learning

RL faces challenges with high-dimensional state and action spaces. When these two spaces are large, the policy's function approximation and exploration become difficult. One way to overcome this difficulty is to (explicitly or implicitly) decompose the action space into multiple levels. This is known as *hierarchical reinforcement learning* (HRL) for action abstraction [45, 52, 211]. In HRL, there are multiple levels of control (usually 2). A set of detailed actions can be learned as an *action primitive* corresponding to a *sub-goal*. The sub-goal can be, e.g., a particular environment state, a particular moving pattern, a latent variable, etc. All the sub-goals form the higher-level action space. The main aim of HRL is to discover reusable action primitives in a task or a set of tasks. By avoiding training lower-level actions repetitively, the effort needed to learn the task can be reduced. A number of previous studies of HRL managed to conquer some environments that are difficult for non-Hierarchical RL frameworks. A detailed review of HRL is provided in Chap. 3.2.

2.1.10 Soft actor-critic

Note: *this subsection reuses Section 3.3 of the thesis author's publication [88] with modifications.*

Soft actor-critic (SAC) is a popular model-free, off-policy deep RL algorithm that uses experience replay, which has been tested on various robotic control tasks and that shows promising performance [83, 84]. A SAC agent learns to maximize reinforcement

returns as well as the entropy of its policy so as to obtain more rewards while keeping actions sufficiently stochastic.

A typical SAC implementation can be described as follows. The state value function $V(s)$, the state-action value function $Q(s,a)$, and the policy function $\pi(a|s)$ are parameterized by neural networks, indicated by ψ, λ, η, respectively. The action of SAC is a continuous variable, obtained by

$$a_\eta(s) = \tanh\left(\mu_\eta(s) + \xi\sigma_\eta(s)\right), \tag{2.19}$$

where the variables $\mu_\eta(s)$ and $\sigma_\eta(s)$ are the actual outputs of the policy network, and ξ is a random variable sampled from a diagonal-covariance unit-Gaussian distribution ($\xi\sigma_\eta(s)$ follows element-wise multiplication). For the critic, SAC used two Q-networks to alleviate the overestimation of TD-error [83]. We indicate the parameters of the two Q-networks as λ_1 and λ_2. Following common practice in deep RL [149], SAC uses a target network for each Q-networks. Each target network has the same structure as the corresponding Q-network but is not trained using RL loss functions. Let $\bar{\lambda}_1, \bar{\lambda}_2$ denote the parameters of the two target networks. At each training step, $\bar{\lambda}_i$ is updated by $\bar{\lambda}_i \leftarrow 0.995\bar{\lambda}_i + 0.005\lambda_i$ $(i = 1, 2)$. Also, an entropy coefficient factor (also known as the temperature parameter), denoted by α, is learned to control the degree of stochasticity of the policy [84]. The parameters are learned by simultaneously minimizing the following loss functions (note that each loss function is minimized w.r.t. the corresponding parameters, e.g., gradient descent only applies to ψ when minimizing $J_V(\psi)$).

$$J_V(\psi) = \mathbb{E}_{s_t\sim\mathcal{B}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\eta}\left[\min_{i=1,2} Q_{\bar{\lambda}_i}(s_t,a_t) - \alpha\log\pi_\eta(a_t|s_t)\right]\right)^2\right], \tag{2.20}$$

$$J_Q(\lambda) = \sum_{i=1,2}\mathbb{E}_{(s_t,a_t)\sim\mathcal{B}}\left[\frac{1}{2}\left(Q_{\lambda_i}(s_t,a_t) - \left(r_t + \gamma\mathbb{E}_{s_{t+1}\sim\mathcal{B}}\left[V_\psi(s_{t+1})\right]\right)\right)^2\right], \tag{2.21}$$

$$J_\pi(\eta) = \mathbb{E}_{s_t\sim\mathcal{B}}\left[\mathbb{E}_{a_\eta(s_t)\sim\pi_\eta(s_t)}\left[\alpha\log\pi_\eta\left(a_\eta(s_t)|s_t\right) - \min_{i=1,2} Q_{\lambda_i}(s_t, a_\eta(s_t))\right]\right], \tag{2.22}$$

$$J(\alpha) = \mathbb{E}_{s_t\sim\mathcal{B}}\left[\mathbb{E}_{a\sim\pi_\eta(s_t)}\left[-\alpha\log\pi_\eta(a|s_t) - \alpha\mathcal{H}_{\text{tar}}\right]\right], \tag{2.23}$$

where $\mathcal{B}$ is the replay buffer from which s_t is sampled, and $\mathcal{H}_{\text{tar}}$ is the target entropy. To compute the gradient of $J_\pi(\eta)$ (Eq. 2.22), SAC assumes a Gaussian distribution of the the reparameterization trick [115] is used on the action $a_\eta(s_t)$. Reparameterization of action is not required in minimizing $J(\alpha)$ (Eq. 2.23) since $\log\pi_\eta(a|s_t)$ does not depends on α.

2.1.11 Twin delayed deep deterministic policy gradient

Twin delayed deep deterministic policy gradient (TD3, proposed by Fujimoto et al. [70]) is another popular model-free actor-critic RL algorithm applicable to off-policy cases. TD3 was proposed at around the same time as SAC [83]. The performance of TD3 was shown to be comparable to SAC in many robotic control tasks [70, 84].

Unlike SAC, TD3 does not consider entropy regularization but optimizes a deterministic policy function (greedy action). Exploration of TD3 agents is realized by

adding external noise to the deterministic policy. The learning of the critic can be considered as Q-learning for continuous action. TD3 also uses two Q-functions and the corresponding target Q-networks as in SAC. Let $\lambda_1, \lambda_2, \bar{\lambda}_1, \bar{\lambda}_2$, and η denote the parameters of the two Q-networks, two target networks, and policy network, respectively. Suppose the replay buffer is $\mathcal{B}$, the loss functions of TD3 are

$$J_Q(\lambda) = \sum_{i=1,2} \mathbb{E}_{(s_t,a_t)\sim\mathcal{B}} \left[\frac{1}{2} \left(Q_{\lambda_i}(s_t, a_t) - \left(r_t + \gamma \min_{j=1,2} Q_{\bar{\lambda}_j}(s_{t+1}, \tilde{a}(s_{t+1})) \right) \right)^2 \right], \quad (2.24)$$

$$J_\pi(\eta) = -\mathbb{E}_{s_t\sim\mathcal{B}} \left[Q_{\lambda_1}(s_t, a_\eta(s_t)) \right], \quad (2.25)$$

where $\tilde{a}(s_{t+1})$ is sampled from the target policy:

$$\tilde{a}(s_{t+1}) = \text{Clip}(a_\eta(s_{t+1}) + \text{Clip}(\epsilon, -c, c), a_{min}, a_{max}), \quad \epsilon \sim \mathcal{N}(0, \sigma), \quad (2.26)$$

where c and σ are hyperparameters. Such clipping essentially acts as a regularization for TD3. It addresses a particular failure without clipping: if an incorrect sharp peak formed in the Q-network for some actions, the policy network might quickly learn to exploit that peak, leading to brittle or incorrect behavior.

2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) are the name for a class of ANNs in which connections between neurons form a directed graph along a discrete-steps sequence [50]. RNNs usually deal with many-to-one or many-to-many mapping from a sequence of input variables to a (sequence of) output variable(s), e.g., when the input is a sentence consisting of a sequence of French words and the output is translated German sentence. RNNs contain some recurrently-connected neurons, which are represented by *internal states* $\boldsymbol{h}_t$. The internal states are usually vectors, denoted by bold symbols. An RNN models the dependence[2] $\boldsymbol{h}_t = g(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$, where t indicates the order of one input in a given sequence $(\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T)$. t will be called *step* in this thesis, though it is not necessarily a time step. An output can be obtained from $\boldsymbol{h}_t$ using an FNN: $\boldsymbol{y}_t = f(\boldsymbol{y}_t)$.

Because RNNs can model contextual (step-to-step) dependence (e.g., $\boldsymbol{h}_t = g(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$), the internal states $\boldsymbol{h}_t$ play a role in memory. The inputs before current step $(\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{t-1})$ are referred to as *historical* inputs, and those after current step $(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \cdots)$ as *future* inputs. Because RNNs can encode information from historical (and future) inputs by the internal states, RNNs are widely used in tasks with contextual dependence, such as natural language processing, speech/music recognition/generation, video processing, etc. In the following sections, we will introduce several types of RNNs involved in the thesis.

2.2.1 Simple RNN

The simplest and earliest RNNs that can be used to solve practical problems were proposed by Elman et al. [50] and Jordan et al. [105] in the 1990s. They both used one hidden layer of $\boldsymbol{h}$ and made self-feedback connections in the network.

[2]For some complex RNNs, the dependence can be more complicated, e.g. $\boldsymbol{h}_t$ can depend on $(\boldsymbol{h}_0, \boldsymbol{h}_1, \cdots, \boldsymbol{h}_{t-1})$ and even $(\boldsymbol{h}_{t+1}, \cdots, \boldsymbol{h}_T)$.

Elman-type RNN [50] can be described as

$$\boldsymbol{h}_t = \sigma_h(W_h \boldsymbol{x}_t + U_h \boldsymbol{h}_{t-1} + b_h), \tag{2.27}$$
$$\boldsymbol{y}_t = \sigma_y(W_y \boldsymbol{h}_t + \boldsymbol{b}_y), \tag{2.28}$$

where $\boldsymbol{h}_t$ and $\boldsymbol{y}_t$ are known as the *hidden state* and *RNN output* at step t, and $\boldsymbol{x}_t$ is the input vector. The weight matrices W_h, W_y, U_h and bias vectors $\boldsymbol{b}_h, b_y$ are trainable parameters of the network. The activation functions σ_h and σ_y can be hyperbolic tangent or sigmoid functions.

Jordan-type RNN [105] is very similar to Elman-type, but the term $U_h h_{t-1}$ is replaced by $U_y \boldsymbol{y}_{t-1}$ in Eq. 2.27.

2.2.2 Continuous-time RNN

The continuous-time recurrent neural network (CTRNN) [71] takes the form of an ordinary differential equation where neuron potentials represent dependent variables. The time evolution of the network is computed using the Euler method, as shown in the following. Let $\boldsymbol{u}_t$ denote the hidden state at step t. The forward dynamics of a CTRNN can be written as

$$\boldsymbol{u}_t = \left(1 - \frac{1}{\tau}\right) \boldsymbol{u}_{t-1} + \frac{1}{\tau} \left(W_{cu} \boldsymbol{c}_{t-1} + W_{xu} \boldsymbol{x}_t + \boldsymbol{b}_u\right), \tag{2.29}$$
$$\boldsymbol{c}_t = \tanh(\boldsymbol{u}_t). \tag{2.30}$$

where τ is the time constant, $\boldsymbol{x}_t$ is the input at step t, and $W, \boldsymbol{b}$ denote the synaptic weights and biases of the corresponding connections.

2.2.3 Multiple-timescale RNN

A *multiple-timescale recurrent neural network* (MTRNN) [248] stacks multiple layers of CTRNN. The time constant of each layer is different, which is used as a biologically plausible model to reflect the fact that multiple timescales are involved in motor learning and development [101, 118, 156, 199]. Various versions of MTRNNs have been applied to continuous sensory-motor studies, such as robot learning [163, 248], sequence prediction [4], and video recognition [31].

Inspired by how our sensory and motor neurons interact with the environment, only the first layer (bottom layer) of the MTRNN receives input (sensory information) and provides output (motor commands). Another feature of MTRNN is that each layer only connects to its adjacent layers, which is necessary for its intrinsic hierarchy [248]. For an L-layers MTRNN, the lth layer has the following forward dynamics (using the same symbols as in Eq. 2.29 with a superscription $l = 1, 2, ..., L$ to indicate the layer):

$$\boldsymbol{u}_t^l = \begin{cases} \left(1 - \frac{1}{\tau^l}\right) \boldsymbol{u}_{t-1}^l + \frac{1}{\tau^l} \left(W_{cu}^{l+1,l} \boldsymbol{c}_{t-1}^{l+1} + W_{cu}^{l,l} \boldsymbol{c}_{t-1}^l + W_{xu} \boldsymbol{x}_t + \boldsymbol{b}_u^l\right) & \text{if } l = 1 \\ \left(1 - \frac{1}{\tau^l}\right) \boldsymbol{u}_{t-1}^l + \frac{1}{\tau^l} \left(W_{cu}^{l-1,l} \boldsymbol{c}_{t-1}^{l-1} + W_{cu}^{l,l} \boldsymbol{c}_{t-1}^l + \boldsymbol{b}_u^l\right) & \text{if } l = L \\ \left(1 - \frac{1}{\tau^l}\right) \boldsymbol{u}_{t-1}^l + \frac{1}{\tau^l} \left(W_{cu}^{l-1,l} \boldsymbol{c}_{t-1}^{l-1} + W_{cu}^{l,l} \boldsymbol{c}_{t-1}^l + W_{cu}^{l+1,l} \boldsymbol{c}_{t-1}^{l+1} + \boldsymbol{b}_u^l\right) & \text{otherwise} \end{cases} \tag{2.31}$$

Note that the hyperparameter τ^l indicates the timescale of changing of hidden state in layer l. Larger τ^l leads to slower dynamics in Eq. 2.31. Inspired by studies of motor learning [101, 118, 156, 199], MTRNN models usually set τ^l larger for higher layers [248].

2.2.4 Gated RNN

Although successful in very simple tasks, plain RNN suffers from the vanishing gradient problem [19, 96, 97], which makes it difficult to deal with long-term dependence. Fortunately, recent advances in RNN have largely improved the problem of gradient vanishing by developing the more powerful *gated RNN* architecture such as *long short-term memory* (LSTM) [98] and *gated recurrent unit* (GRU) [29]. LSTM and GRU are quite similar in terms of designing ideas. They employ non-linear gating units instead of the conventional hyperbolic tangent activation function to regularize the amplitude of gradients so as to overcome the gradient vanishing problem. It is reported that there is no significant performance difference between LSTM and GRU in practice [33].

Since we will use LSTM in the later parts of the thesis, the dynamics of an LSTM network [98] is shown below.

$$\text{forget gate's activation vector } \boldsymbol{f}_t = \sigma_g(W_f \boldsymbol{x}_t + U_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \tag{2.32}$$

$$\text{input gate's activation vector } \boldsymbol{i}_t = \sigma_g(W_i \boldsymbol{x}_t + U_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \tag{2.33}$$

$$\text{output gate's activation vector } \boldsymbol{o}_t = \sigma_g(W_o \boldsymbol{x}_t + U_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \tag{2.34}$$

$$\text{cell input activation vector } \tilde{\boldsymbol{c}}_t = \sigma_c(W_c \boldsymbol{x}_t + U_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \tag{2.35}$$

$$\text{cell state vector } \boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tilde{\boldsymbol{c}}_t, \tag{2.36}$$

$$\text{hidden state vector (LSTM output) } \boldsymbol{h}_t = \boldsymbol{o}_t \circ \sigma_c(\boldsymbol{c}_t), \tag{2.37}$$

where $\boldsymbol{x}_t$ is the input vector at step t, and $\circ$ denotes element-wise product. For $p = f, i, o, c$: W_p, U_p are the weight matrices and b_p is the bias vector for each activation vector p. σ_g and σ_c are the sigmoid function and the hyperbolic tangent function, respectively. A gated RNN can be intuitively understood as an adaptive-timescale version of the CTRNN (with some more parameters to increase the expressive power), where the time constant is decided by functions of the hidden state and input.

2.2.5 Variational RNN

Note: *this subsection reuses Section 3.3 of the thesis author's publication [88] with modifications.*

All the previously introduced RNNs contain only deterministic computation nodes. However, in many scenarios, we need to tackle probabilistic latent variables, and most of the distributions are intractable. A powerful tool to approximate such probabilistic variables is *variational Bayes* [115], which is usually modeled by a graph (such as a neural network) containing many stochastic nodes. Each stochastic node can be expressed by a simple, analytic output distribution, e.g., normal distribution, given an input.

The *variational recurrent neural network* (VRNN) [34] was developed as a recurrent extension of the variational auto-encoder (VAE, [115]), composed of a variational

generation model and a variational inference model. It is a recurrent latent variable model that can learn to encode and predict complicated sequential observations $\boldsymbol{x}_t$ with a stochastic latent variable $\boldsymbol{z}_t$.

The generation model predicts future observations given its internal states,

$$\begin{aligned} \boldsymbol{z}_t &\sim \mathcal{N}\left(\boldsymbol{\mu}_{p,t}, \mathrm{diag}(\boldsymbol{\sigma}_{p,t}^2)\right), \quad \left[\boldsymbol{\mu}_{p,t}, \boldsymbol{\sigma}_{p,t}^2\right] = f^{\mathrm{prior}}(\boldsymbol{d}_{t-1}), \\ \boldsymbol{x}_t|\boldsymbol{z}_t &\sim \mathcal{N}\left(\boldsymbol{\mu}_{y,t}, \mathrm{diag}(\boldsymbol{\sigma}_{y,t}^2)\right), \quad \left[\boldsymbol{\mu}_{y,t}, \boldsymbol{\sigma}_{y,t}^2\right] = f^{\mathrm{decoder}}(\boldsymbol{z}_t, \boldsymbol{d}_{t-1}), \end{aligned} \tag{2.38}$$

where fs are parameterized mappings, such as feedforward neural networks, and $\boldsymbol{d}_t$ is the state variable of the RNN, which is recurrently updated by

$$\boldsymbol{d}_t = f^{\mathrm{RNN}}(\boldsymbol{d}_{t-1}; \boldsymbol{z}_t, \boldsymbol{x}_t). \tag{2.39}$$

The inference model approximates the latent variable $\boldsymbol{z}_t$ given $\boldsymbol{x}_t$ and $\boldsymbol{d}_t$.

$$\boldsymbol{z}_t|\boldsymbol{x}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{z,t}, \mathrm{diag}(\boldsymbol{\sigma}_{z,t}^2)\right), \text{ where } \left[\boldsymbol{\mu}_{z,t}, \boldsymbol{\sigma}_{z,t}^2\right] = f^{\mathrm{encoder}}(\boldsymbol{x}_t, \boldsymbol{d}_{t-1}). \tag{2.40}$$

For sequential data that contain T time steps, learning is conducted by maximizing the evidence lower bound $ELBO$, like that in a VEA [115], where

$$\begin{aligned} ELBO = \sum_t^T \big[& -D_{KL}(q(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{x}_{1:t})||p(\boldsymbol{z}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{x}_{1:t-1}))\big] \\ & + \mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{x}_{1:t}, \boldsymbol{z}_{1:t-1})}\left[\log\left(p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t}, \boldsymbol{x}_{1:t-1})\right)\right], \end{aligned} \tag{2.41}$$

where p and q are parameterized PDFs of $\boldsymbol{z}_t$ by the generative model and the inference model, respectively.

Despite the aforementioned VRNN model, there are also other ways to apply variational Bayes on RNN. For example, predictive coding-inspired VRNN [5] applied predictive coding [170] on a variational RNN model, in which the posterior distribution of $\boldsymbol{z}$ is updated to minimize the prediction error during online operation.

Chapter 3

Self-Organization of Action Hierarchy

Note: *This chapter reused the thesis author's publication [87] (the other two authors of the paper are the thesis author's supervisor and co-supervisor) with modification and re-organization to fit the thesis.*

- Dongqi Han, Kenji Doya, and Jun Tani. "Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks." *Neural Networks*, 129:149–162, 2020.

3.1 Background

RL has been successfully applied to a large variety of tasks, such as robotic control [83, 133], game playing [148, 149], neural architecture design [256], etc. Researchers have improved both the theory and implementation of RL algorithms and methods for more efficient and robust learning. From the theory side, for example, Degris et al. [41] generalized policy gradient to off-policy case, and Munos et al. [153] proposed new operators with a good convergence property for multi-steps looking ahead RL. As for implementation, Fortunato et al. [61] proposed noisy network models for facilitating exploration in RL, and Vieillard et al. [233] discussed a regularization of loss functions in RL. These improvements are too many to list, leading to the huge success of deep RL in single-task learning.

However, it was relatively under-explored for RL studies involving skill transfer or sharing from task to task, usually called transfer-RL [220] or meta-RL [182, 225, 238, 239]. It is known that humans perform much more efficient transfer learning across similar tasks, while only recently have RL studies started to consider approaches of transfer-RL or meta-RL in relatively challenging tasks [1, 57, 58, 138, 178, 221, 238]. Nonetheless, these methods usually require an agent to alternatively interact with a set of similar environments so that its (meta) policy can be optimized over all the environments [57, 239]. However, in the real world, humans can learn much more efficiently to solve an unseen, novel task, which is different but similar to a previously learned one, by autonomously recognizing the skills that can be reused [238]. In other words, transfer learning and meta-learning in the brain can be performed in an end-to-end manner, whilst conventional machine learning agents normally require providing explicitly or implicitly task similarity [239].

A promising RL scheme for skill reuse/transfer is hierarchical RL [45, 210], which divides action into multiple levels (usually 2 levels), and a higher-level action (also known as an *option* [10, 210], a *skill* [190], or a *sub-goal* [8] in a more general manner). The detailed actions under a given higher-level action is called an *action primitive*, which is expected to fulfill a sub-task. By learning the detailed control policy of each action primitive, an agent should be able to efficiently relearn a new task which is re-composition of learned action primitives by avoiding repetitively learning the low-level control policy. More discussion about prior work on hierarchical RL can be found in Sec. 3.2.

In previous studies by Sharma et al. [190] and Xu et al. [247], it has been investigated how to use the action primitives acquired via an unsupervised learning task to solve a new RL task. However, a less touched problem is how the discovered action primitives can be taken advantage of to perform transfer learning in an end-to-end manner. While sometimes directly reusing learned action primitives is more efficient, humans and animals are also able to re-adapt the lower-level maneuver when necessary. How can this be done in artificial agents by learning through self-exploration? We seek inspiration from the brain and propose a novel multi-timescale RNN architecture and an off-policy actor-critic algorithm for RL. 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 [25, 101, 154, 156, 177, 199], as well as in the discount factors across different levels in an RNN. Although it has been shown that the introduction of multiple-timescale neural activation dynamics in RNNs enhances the development of hierarchy in supervised learning [248], such a possibility in RL remains to be investigated. In most RL algorithms, the discount factor (for an MDP) is treated as a single hyperparameter. However, the experimental studies by Enomoto et al. [51] and Tanaka et al. [214, 215] 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 [14, 15, 91, 201]. Chung et al. [34] and Fraccaro et al. [62] have shown that various types of stochastic RNN models can learn to extract probabilistic structures hidden in temporal patterns by using variational Bayes approaches in supervised learning. For deep RL, the stochasticity in the hidden states of a policy network should facilitate abstracted-level motor exploration (in comparison to conventional RL methods using only noisy motor actions) if the network has acquired a representation

of abstracted-level actions. It was shown that stochastic FNNs facilitate efficient exploration and improve performance in RL [60, 61]. Therefore, we are interested in whether and how stochastic RNNs promote the exploration and extraction of task features.

The key contribution of the proposed work is a novel RL framework, ReMASTER, which is characterized by a multiple-timescale, stochastic RNN architecture. ReMASTER integrates these two essential ideas (multiple-timescale property and neuronal stochasticity) with off-policy actor-critic algorithms, in a model-free manner. Note that ReMASTER is not a policy evaluation or policy improvement algorithm but a general framework that can corporate with any off-policy actor-critic algorithm.

We first 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 a 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 the 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 pursue an undesired sub-goal by clamping high-level RNN states, analogous to animal optogenetic experiments [150, 169].

We then examined the 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 previously-undertaken 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 to both multiple timescales and neuronal stochasticity used in the model. Furthermore, we scaled ReMASTER up to more challenging tasks by incorporating recently developed RL algorithms. We showed that ReMASTER outperformed the alternative models in more real-world tasks with high-dimensional observation and action spaces.

The rest of this chapter is arranged as follows. Chap. 3.2 provides a review of previous work on hierarchical RL. Chap. 3.3 details the RNN model used in ReMASTER. We then conducted a comprehensive empirical investigation of ReMASTER using a two-wheeled robot navigation task containing a sequence of 3 sub-tasks, followed by corresponding results (Chap. 3.4)). Moreover, we demonstrate the scalability of ReMASTER using several kinds of additional HRL tasks with the challenge of larger state and action spaces in Chap. 3.5. Chap. 3.6 summarizes how ReMASTER distinguishes itself from prior approaches and concludes the results. Finally, we discuss possible connections to neuroscience in Chap. 3.7.

3.2 Prior Work on Hierarchical RL

This section provides a review of previous methods on hierarchical RL (HRL). In particular, we focus on the recent studies using deep RL approaches. Since a detailed explanation of ReMASTER is essential before understanding its novelties, we will detail

how ReMASTER distinguishes itself from the existing approaches later in Chap. 3.6.

Before deep learning techniques were widely used in RL [149, 196], some ideas of HRL[1] [52] had already been formed in the 1990s [39, 45, 165, 210]. For example, Dayan & Hinton [39] proposed the Feudal RL framework, in which the task is manually divided into multiple levels like a feudal society. The controller at each level only cares about the task of the current level and leaves the detailed task to the lower-level sub-controller. Parr & Russel [165] introduced the "hierarchy of machines" for RL, in which a state machine, designed by the experimenter's prior knowledge about the task, was used to reduce the search space. Another HRL method, known as MAXQ value decomposition [45], decomposes the original MDP into a hierarchy of smaller MDPs. Correspondingly, the value function is also decomposed as an additive combination of the value functions of the smaller MDPs. The option framework, proposed by Sutton et al. [210], systematically introduced the idea of temporally abstracted actions. More specifically, "options" are defined as discrete labels representing the sub-tasks, with a sub-policy defined for each option. An option can be executed for multiple consecutive steps (it can be learned when to stop), and the sub-policy of each option can be learned using standard RL methods such as Q-learning [241] with minor modifications. While these pioneering studies were limited to grid worlds or simple control tasks, they provided the theoretical and practical foundations for the follow-up work on HRL. However, these early ideas all require prior knowledge about the task hierarchy or information about the sub-goals.

Researchers have recently started to seek deep RL methods to acquire hierarchical policies relying less on the prior knowledge. Among the recent deep HRL studies, a popular category is based on the option framework [10, 11, 130, 174, 211, 221]. The option-critic architecture [10] extended the original option framework [211] to a 2-level deep HRL architecture that requires no prior knowledge about the task hierarchy (However, the number of options needs to be pre-defined in an *ad-hoc* manner). The option-critic architecture trains the low- and high-level Q-functions with Q-learning [241] and the termination function for the high level with policy gradient. Later, the option-critic architecture was generalized to arbitrary levels of hierarchy by Riemer et al. [174]. Moreover, Tessler et al. [221] proposed an HRL approach to perform life-long learning, where the authors first pre-trained the lower-level policies using pre-defined sub-tasks and then trained the higher-level policy model with fixed lower-level policies. Another HRL method introduced by Li et al. [130] did not learn the termination function; instead, each lower-level policy is carried out for n steps, where n is a random positive integer in a pre-defined range. The 2-levels of policies are trained simultaneously using their approximated policy gradient algorithm. It is worth mentioning that augmenting the number of options during learning is also possible: Bargaria & Konidaris [11] proposed a method to learn options with skill chaining [117], where each option represents a sub-path, many of which chain together to achieve the goal. For the aforementioned methods [10, 11, 130, 174, 211, 221], the options are discrete (thus, there are multiple lower-level actors). Each option is usually executed for multiple time steps, and the higher-level controller must be trained or pre-defined to

[1] Although state abstraction [44, 127] is arguably a part of HRL, the current discussion restricts itself to action abstraction.

determine when to terminate an option. In this work, we investigate a different scheme: the higher-level action as a continuous variable that may change at every step.

Another family of HRL approaches uses the sub-goal state/observation as the label for a sub-policy [8, 129, 180, 180], among which a representative is hindsight experience replay (HER) [8]. HER includes both the current and goal state as the input of a value function [180]. HER alleviates the difficulty of sparse rewards by augmenting the original task with additional pseudo tasks, where the reached states from the agent's experiences are used as pseudo goals. Levy et al. [129] improved HER using additional penalties to avoid unreachable sub-goals and demonstrated effective learning using a hierarchical architecture with 3 levels. Although the sub-goal-based approaches proved to solve many tasks that are considered hard for vanilla RL, they require the environment always to provide a goal state. Thus HER cannot be directly generalized to the task without a detailed goal state, such as a board or card game in which there are various states to win. By contrast, we aim to develop a general RL framework that does not need an explicit goal state.

There are also HRL methods considering the change of state as a sub-goals. The Feudal network for HRL [232], which was inspired by the original feudal RL [39], proposed a 2-levels architecture for HRL. The raw observation was embedded into a latent space s. The high-level action reflects the change of s, which was learned to solve the original task, while the low-level policy learned to fulfill this change. Both levels were trained with on-policy RL [148] to avoid representation shift (i.e., the distribution of state transitions in the replay buffer changes with updating the policy). Similar to the Feudal network for HRL, Nachum et al. [155] proposed an HRL method to take advantage of experience replay by introducing an off-policy correction for the higher level. It was shown that straightforwardly using the change of raw observation as high-level action yields better performance. However, we aim to investigate what contributes to the self-organization of an action hierarchy via end-to-end RL, where no explicit meaning is assigned to the higher-level action. Thus, our work is different from the aforementioned ones [155, 232], in which the higher-level action was designed to reflect the change of state from one step to the next.

A class of methods also aims to discover action primitives in RL by optimizing an unsupervised/self-supervised objective function, usually based on information theory [3, 53, 79, 190, 247]. Here we refer to them as *variational skill discovery* (VSD) methods [3]. These methods employ a latent variable z to label an action primitive, where z can be discrete [3, 53, 79] or continuous [190, 247]. Then, pseudo (intrinsic) rewards are designed to train the policy model $\pi(a|z, s)$ with RL, where a is action, and s is state. The pseudo rewards reflect an information-theoretic objective that encourages the skills to be diverse and predictable by states, such as the variational lower bound of the mutual information between the set of skills and skill termination states [79]. While the VSD methods are powerful in discovering action primitives in many challenging robotic locomotion tasks, the action primitives cannot be called "self-organized" due to the additional learning objective designed for this. Also, the VSD methods have some limitations, e.g., implicitly assuming the smoothness of observable states (thus, they were not shown to be effective on tasks with image observations). Our work differs from the VSD methods in that ReMASTER trains the model using only the environmental reward with RL. Also, ReMASTER aims to learn an HRL task end-to-end, while the

VSD methods focus on learning the action primitives [3, 53, 79, 190].

Moreover, some studies consider the action hierarchy in meta-RL (i.e., learning to solve a set of tasks that share some common properties [182, 225, 238, 239]), among which the studies by Florensa et al. [60] and Frans et al. [63] are related to ReMASTER. Florensa and colleagues [60] proposed a stochastic network for HRL. However, the low-level policy needs to be pre-trained using pseudo rewards and fixed when training the high-level policy, which is not end-to-end. ReMASTER shared a similar idea with the work by Frans et al. [63]: the high-level and low-level networks optimize RL objectives with different timescales. However, a key difference is that Frans et al.'s work [63] did not apply intrinsic constraint of temporal dynamics to each level of the neural network, which is an essential feature of ReMASTER. Nonetheless, we consider a standard POMDP setting without assuming the agent always has access to a set of tasks as in meta-RL. Besides, there also exist HRL methods in offline RL/RL from demonstration [189] or model-based RL [132, 246]. These works are beyond the scope of our study that considers model-free, online RL in POMDPs.

To summarize, HRL methods can be divided into two categories, which I refer to as temporally extended and latent-sub-goal HRL, respectively. The temporally extended HRL methods used discrete labels for action primitives. Each higher-level action corresponds to one sub-policy model, and each sub-policy, when selected, will be executed for multiple steps [10, 11, 39, 45, 60, 63, 121, 130, 132, 165, 174, 210, 221, 246, 251]. The latent-sub-goal HRL methods model the label of the current action primitive as a latent variable (usually continuous), which is included as an input argument to the low-level policy model[3, 8, 53, 79, 129, 155, 180, 189, 190, 232, 247]. Our work belongs to the latent-sub-goal HRL category. However, ReMASTER is not an incremental method of the existing ones. We will specify the novelties of ReMASTER in Chap. 3.6 after fully introducing ReMASTER in the subsequent sections.

3.3 ReMASTER

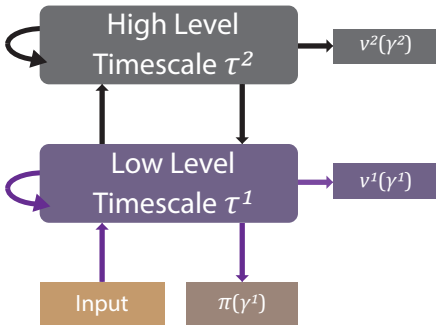


Figure 3.1: The basic structure of MTSRNN is shown for the case of a 2-level configuration used in this work. However, additional levels can readily be stacked onto it.

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. 3.1 shows

the case of a 2-level MTSRNN where γ^l represents the characteristic discount factor at l-th level. v^l as the value function at l-th level can be learned by any policy evaluation algorithm [208] using the corresponding γ^l (Although we show the state value function v here, we can also use the state-action value function q by adding action a as input to it). 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, the policy is improved to maximize expected return w.r.t. the lowest discounter factor γ^1.

3.3.1 Multiple timescale stochastic RNN

Here we describe detailed mechanisms of an 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[2], we have

$$\begin{aligned}\boldsymbol{u}^l(t) =&(1-\frac{1}{\tau^l})\boldsymbol{u}^l(t-1)+\frac{1}{\tau^l}\left[W_{cu}^{l-1,l}\boldsymbol{c}^{l-1}(t)+\right.\\ &\left.W_{cu}^{l,l}\boldsymbol{c}^l(t-1)+W_{cu}^{l+1,l}\boldsymbol{c}^{l+1}(t-1)+\boldsymbol{b}_u^l\right], \qquad (3.1)\\ \boldsymbol{c}^l(t) =&\tanh\left(\boldsymbol{u}^l(t)+\boldsymbol{\epsilon}^l\boldsymbol{\sigma}^l(t)\right), \qquad (3.2)\end{aligned}$$

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 hyperparameter 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 [115]. The hyperparameter 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.

3.3.2 Reinforcement learning

In general, ReMASTER is flexible to the choice of RL algorithms. In this work, we focus on a model-free RL scenario with robotic control tasks with continuous action space. For sample efficiency, we employ off-policy actor-critic algorithms [41, 70, 84] for the experiments using ReMASTER.

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, π 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.

Algorithm 3 summarizes how a ReMASTER agent performs RL in general.

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

Algorithm 3 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 from $\mathcal{B}$
      Compute the loss of critic for all levels
      Compute the loss of actor for the lowest level
    end if
    $t \leftarrow t + 1$
  end while
until training stopped

3.3.3 Experience replay with RNN

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 (Eq. 3.9). 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.

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

- **Recording the RNN states at each step for the initial states in experience replay.** When the agent interacts with the environment, its RNN states at each step are also recorded in the replay buffer. When we perform experience replay with a sequence of state transitions sampled from the buffer, we can use the RNN states recorded at the step right before this sequence as the initial RNN states and then process the sequence with the RNN's forward dynamics. Then we can train the network using BPTT. Despite the simplicity of this approach, it is unclear what algorithmic issues will arise due to the difference between old internal representations and new ones.

- **Using an entire episode as a sequence.** This was used, e.g., in the work by Mnih and colleagues [148], 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 for experiments in Atari Games [92]. Unfortunately, this implementation prevents learning long-term dependence because of the mismatch of initial states, as argued by Kapturowski et al. [110].

- **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.'s study [110], 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 before the start of a sampled sequence. Although remarkable performance on many RL tasks was demonstrated using this approach [110], 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 better decide initial states still remains a challenge for RL with experience replay using RNNs.

3.3.4 Motor and neuronal noise

For continuous sensory-motor tasks, the range of state space can be very large. To enhance the efficiency of motor exploration, we used motor noise generated by the *Ornstein-Uhlenbeck process* [227] (OU-process), like that used in the deep deterministic policy gradient algorithm [133]. The OU process generates temporally auto-correlated noise; thus, the exploration range can be increased with the "inertia" of the noise.

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

$$x_t = -\theta_a x_{t-1} + e\sqrt{2\theta_a}\epsilon_t, \tag{3.3}$$

where $\theta_a = 0.3$ for all of our experiments, and ϵ_t is a white noise with unit Gaussian distribution. e indicates the scale of motor noise, i.e., its standard deviation. Since we dealt with the tasks with sparse rewards provided only when a sub-goal is achieved, such a temporally correlated noise should increase the possibility of achieving a sub-goal by random exploration [133], while the standard deviation of the noise was not too large.

However, it is not necessary to apply temporally correlated noises to hidden states of the MTSRNN since recurrent connections in an RNN already generate the temporal correlation. Thus, we simply applied Gaussian white noises to the neural dynamics as in Eq. 3.2

The scales of motor and neuronal noise will be detailed in Chap. 3.4 and 3.5 respectively.

3.4 Self-organization of action hierarchy using Re-MASTER

3.4.1 Task settings

We propose a sensory-motor task, inspired by the work by Utsunomiya & Shibata [228], referred to as a "sequential target-reaching task". As illustrated in Fig. 3.2(b), a two-wheel robot is required to approach three targets in a sequence. The targets and the robot are located on a 2-dimensional field. 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. 3.2(a). At each step, the action is given as the rotations of two wheels, which are continuous in range $[-180°, 180°]$. The 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, the positions of three targets are randomly set inside the center 8×8 area. The distance between two targets is ensured to be larger than 2. The observation is a 12-D real number vector: $(e^{-d_{red}/5}, e^{-d_{green}/5}, e^{-d_{blue}/5}, e^{-d_{frontwall}/5}, e^{-d_{backwall}/5}, r, \sin\theta_{red}, \cos\theta_{red}, \sin\theta_{green}, \cos\theta_{green}, \sin\theta_{blue}, \cos\theta_{blue})$, where r is the immediate reward at current time step, and other quantities are shown in Fig. 3.2(a).

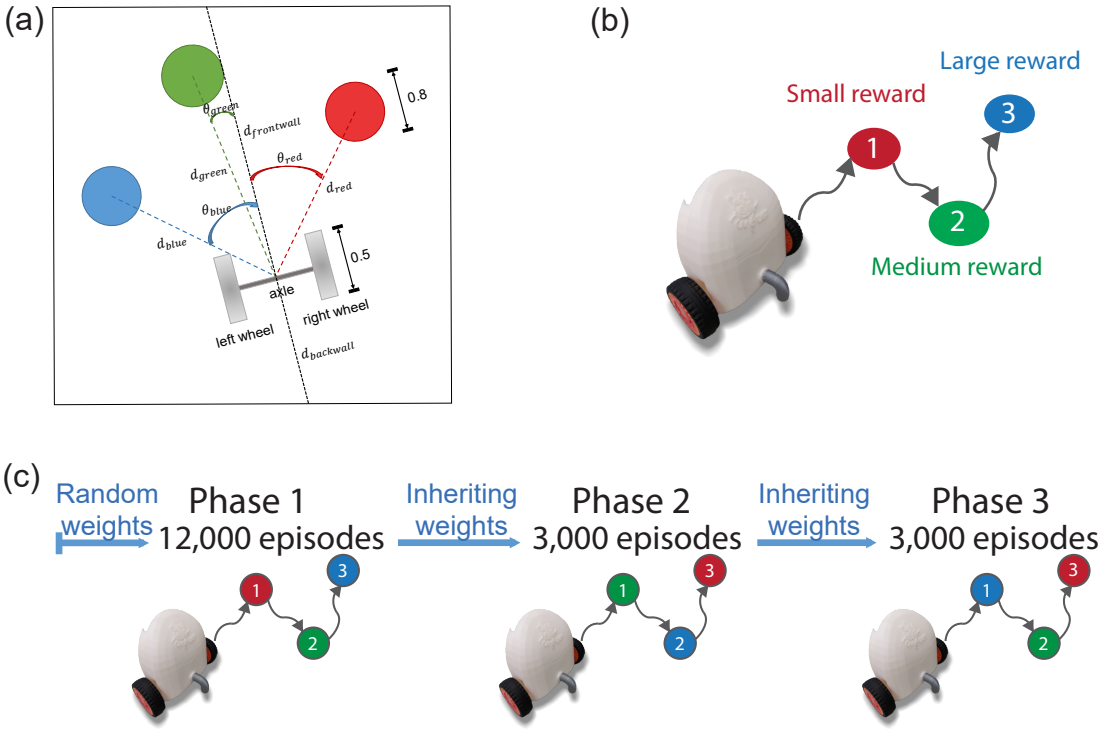


Figure 3.2: Task configurations (**a**) Top view of the task field. The size of the square field is 15×15. Objects are zoomed out for visual clarity. (**b**) The sequential target-reaching task. (**c**) The consecutive relearning task. The positions of the robot and the targets were randomly initialized in each episode.

The sequential target-reaching task is of particular interest because it abstracts many real-world tasks in complicated environments, which involve decomposing a whole task into sub-tasks and executing 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 the development of hierarchical control of actions. Lower levels acquire the control policy for action primitives, while higher levels learn to dispatch those action primitives in a specified sequence.

The robot must reach the three targets in the red-green-blue sequence 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

$$r(t) = 0.8/(1 + d_{red}(t)). \tag{3.4}$$

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

$$r(t) = 2.0/(1 + d_{green}(t)). \tag{3.5}$$

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

$$r(t) = 5.0/(1 + d_{green}(t)) \quad \text{(task done)}. \tag{3.6}$$

In other words, when the robot reaches a target in the correct sequence, it receives a **one-step reward**. The reward is given only if the agent follows the proper sequence. And the reward signal is given only once for each target. If the robot hits the walls, a negative reward -0.1 is given. Otherwise, the reward is zero. 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" and to recognize the correct sub-goal (which can be considered as approaching a target in this task).

Moreover, we propose an extended version of the sequential target-reaching task to examine the transfer learning properties of ReMASTER, referred to as an "consecutive relearning task" (Fig. 3.2(c)). 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 3,000 episodes in phase 2 or 3.

3.4.2 Off-policy advantage actor-critic

Here we detail the actor-critic algorithm used for the sequential target-reaching and consecutive relearning tasks. For the critic, we used an off-policy version of the temporal difference (TD) learning algorithm to train value functions of all levels [41, 208]. Knowing that (i) each level has a characteristic timescale τ^l; (ii) $1/(1-\gamma)$ indicates the eigen-timescale of discounting [46], it is natural to set the values of discount factors as

$$\gamma^l = 1 - \frac{K}{\tau^l}, \tag{3.7}$$

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,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_v \frac{1}{L}\frac{1}{N}\sum_{l}^{L}\sum_{i}^{N}\left[\rho_i \delta_i^l \nabla_{\boldsymbol{\theta}} v^l(\boldsymbol{s}_{0:t_i};\boldsymbol{\theta})\right], \tag{3.8}$$

where we have

$$\rho_i = \frac{\pi(\boldsymbol{a}_{t_i}|\boldsymbol{s}_{0:t_i};\boldsymbol{\theta})}{\pi_{t_i}} \tag{3.9}$$

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

$$\delta_i^l = r_{t_i} + \gamma^l v^l(\boldsymbol{s}_{0:t_i+1};\boldsymbol{\theta}) - v^l(\boldsymbol{s}_{0:t_i};\boldsymbol{\theta}) \tag{3.10}$$

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.

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 $\boldsymbol{\pi}(t) \sim \mathcal{N}\left(\boldsymbol{p}(t), \boldsymbol{e}(t)\right)$, where $\boldsymbol{p}(t) = \tanh\left(W_{ca}\boldsymbol{c}^1(t) + \boldsymbol{b}_{ca}\right)$ is the expected action and $\boldsymbol{e}(t)$ is the exploration noise scale (Chap. 3.4.13). The action is clipped into [-1, 1] since the robot has a maximum speed.

To update the policy function, an advantage policy gradient algorithm was used in an off-policy manner [41] (with a correction by the important sampling ratio ρ_i), where the advantage was estimated by 1-step TD error with discount factor γ^1.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_a \frac{1}{N}\sum_{i}^{N}\left[\rho_i \delta_i^1 \nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi}(\boldsymbol{a}_{t_i}|\boldsymbol{s}_{0:t_i};\boldsymbol{\theta})\right], \tag{3.11}$$

where α_a is the learning rate for the actor.

3.4.3 Noise scales

For exploration, the motor noise scale e (Eq. 3.3) was annealed exponentially w.r.t. episodes, with a minimum value of 0.1:

$$e = 180^\circ \times \left[0.75 \times \exp(-\frac{1}{3000} \times \text{episode}) + 0.1\right], \tag{3.12}$$

where 180° corresponds to the rotation of the two wheels as motor actions.

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

$$\sigma = \sigma_0 \exp(-\frac{1}{3000} \times \text{episode}). \tag{3.13}$$

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

$$e = 180^\circ \times \left[0.75 \times \exp(-\frac{1}{750} \times \text{episode}) + 0.1\right], \tag{3.14}$$

and

$$\sigma = \sigma_0 \exp(-\frac{1}{750} \times \text{episode}). \tag{3.15}$$

For the neuronal noise, we used $\sigma_0 = 0.2$ if not specified, which lead to better results (see Chap. 3.4.13).

3.4.4 Hyperparameters

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 all the tasks.

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 the actor. We used a replay buffer of a maximum size of 500,000 and performed experience replay every 2 steps, using a mini-batch containing 16 sequences with length 25, randomly sampled from the buffer.

We summarize the hyperparameters used for the sequential target reaching task in Table 3.1. Some of the hyperparameters were obtained by random search (see Chap. 3.4.11), and the others are hand-tuned. However, a different choice of hyperparameters will not significantly change our main conclusions (Chap. 3.4.11).

3.4.5 Sequential target-reaching task results

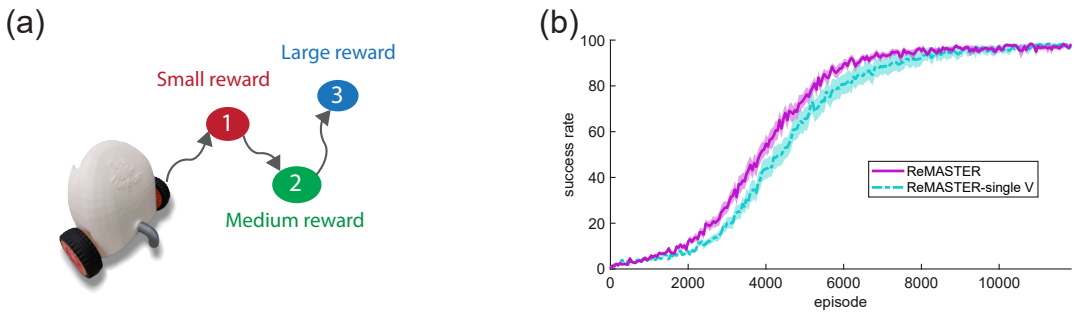


Figure 3.3: The sequential target-reaching task: (**a**) Illustration of the task. (**b**) Performance curve indicated by success rate, where a "success" is defined as finishing the task by reaching all the 3 targets within 50 steps. ReMASTER-single V is the case in which the higher-level value function was not learned. Data are Mean ± S.E.M., obtained from 20 repeats.

Table 3.1: Hyperparameters we used in the sequential goal reaching task and the consecutive relearning task for ReMASTER. The Hyperparameters were obtained by random search or hand-tuned.

Hyperparameter	Description	Value
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
N^2	Number of neurons in the higher level	50
buffer_size	Number of steps recorded in memory	5e5
σ_0	Initial scale of neuronal noise	0.2
n_update	Number of steps per update	2
lr_critic	Learning rate of critic	3e-4
lr_actor	Learning rate of actor	1e-4
α	Decay of the RMSProp optimizers	0.99
batch_size	Number of training sequences.	16
L	Sequence length for truncated BPTT	25

We examined ReMASTER in the sequential target-reaching task. We define the performance measurement using "success rate": an episode is considered successful when the agent completed the task within 50 steps[3] (For reference, the best agent could complete the task with about 20 steps on average). Fig. 3.3(b) shows that ReMASTER can successfully solve this task through self-exploration, achieving more than 95% success rate on average after training. We also tested the case in which the higher-level value function v^2 was not trained. (ReMASTER-single V in Fig. 3.3(b)), which achieved a 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. 3.4(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. 3.4(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. 3.4(a). In contrast, this feature was less obvious in the lower level (second row of Fig. 3.4(a)). Although we only show one agent here, this result is statistically significant (Chap. 3.4.8), and more examples can be found in Fig. 3.6. This result suggests that an MTSRNN with slower dynamics in the higher level enhanced the development of a consistent representation, accounting for a given sub-goal structure through abstraction in the higher level, whereas the lower level dealt with

[3]We employ success rate to evaluate performance because other kinds of measurements have some defects. For example, the total rewards in an episode cannot reflect whether the agent finishes the task in a timely manner since the amount of rewards when reaching the targets is Markovian. Also, computing the average time steps to finish the task is non-trivial because the agent often cannot finish the task within the time step limitation of an episode (before being well trained).

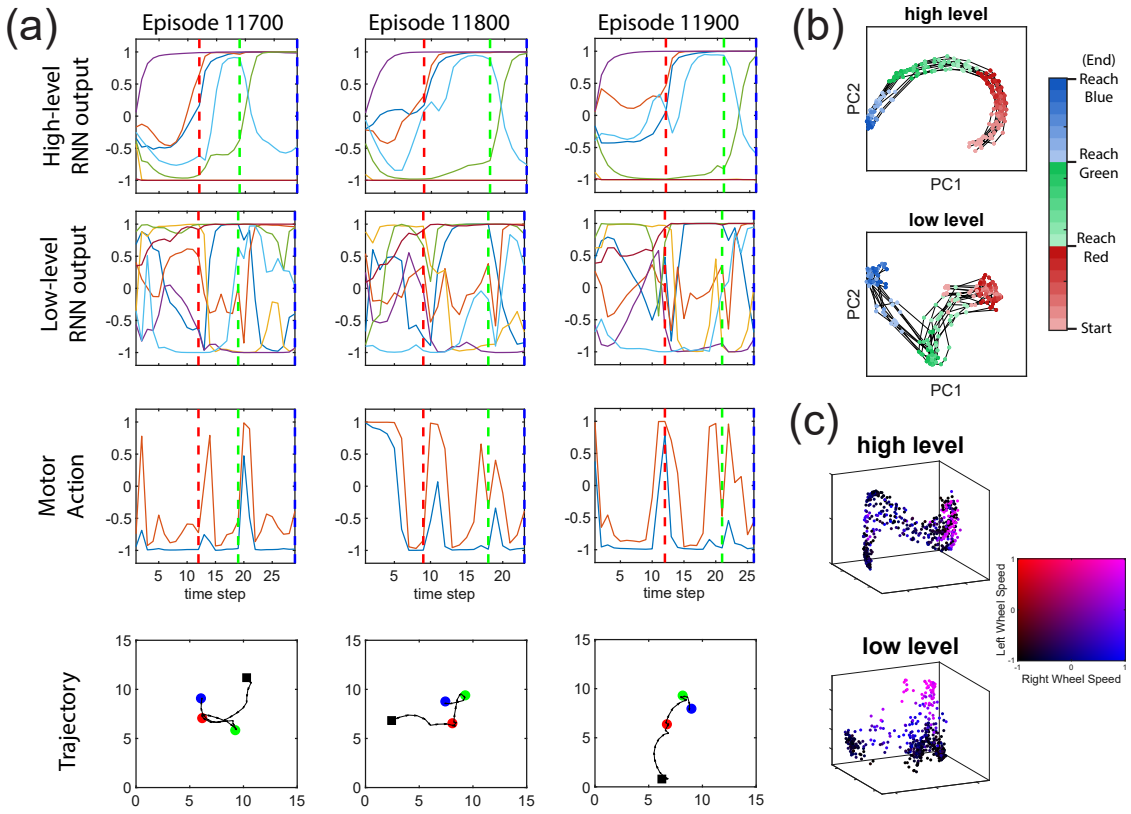


Figure 3.4: Analysis of the sequential target-reaching task using ReMASTER. (**a**) Three example episodes showing the behavior of a well-trained ReMASTER agent. The first and second rows show RNN output $\boldsymbol{c}_t^l$ of two levels, where the vertical, dashed lines indicate the agent's reaching 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 $\boldsymbol{c}_t^l$, using data of the same agent in (a) in episodes 11000, 11100, ..., 11900 (after convergence). Colors mean the agent is approaching the corresponding targets, whereas 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 the speed of the two wheels (see the colormap). The first 3 PCs were plotted.

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. 3.4(b)). We can see that the high-level RNN outputs showed a consistent, sequence-like representation of sub-goals accounted for by its slower dynamics. In contrast, 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 the representation of low-level motor actions (Fig. 3.4(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. The following sections will provide more abundant empirical evidence to support the emergent action hierarchy using ReMASTER.

3.4.6 Consecutive relearning task results

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

By considering this, we carried out experiments in the consecutive relearning task (Fig. 3.5(a)). The experiments were conducted in a lifelong learning manner [195, 224]. We maintained the same learning algorithm and hyperparameters throughout all 3 phases. Synaptic weights were continuously updated without resetting throughout the experiment. At the beginning of each phase, the motor and neuronal noise scales were reset, and the replay buffer was cleared. We also compared performance using ReMASTER to two alternatives 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. 3.5(b-d)). Another alternative used the same algorithm, but replaced the MTSRNN with a single-layer LSTM (LSTM in Fig. 3.5(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 contained 75 cells such that the number of parameters is similar to that of the MTSRNN.

Results are illustrated in Fig. 3.5(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. 3.5(c,d)) starting with previously trained synaptic weights achieved much better sample efficiency than the case of phase 1, which was done from scratch. 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. Note that there was no immediate transfer learning for all the models, i.e., the success rates at the beginning of the relearning phases were nearly

[4]Although we used tuned hyperparameters for better performance, these conclusions indeed hold for a different choice of hyperparameters (Chap. 3.4.11)

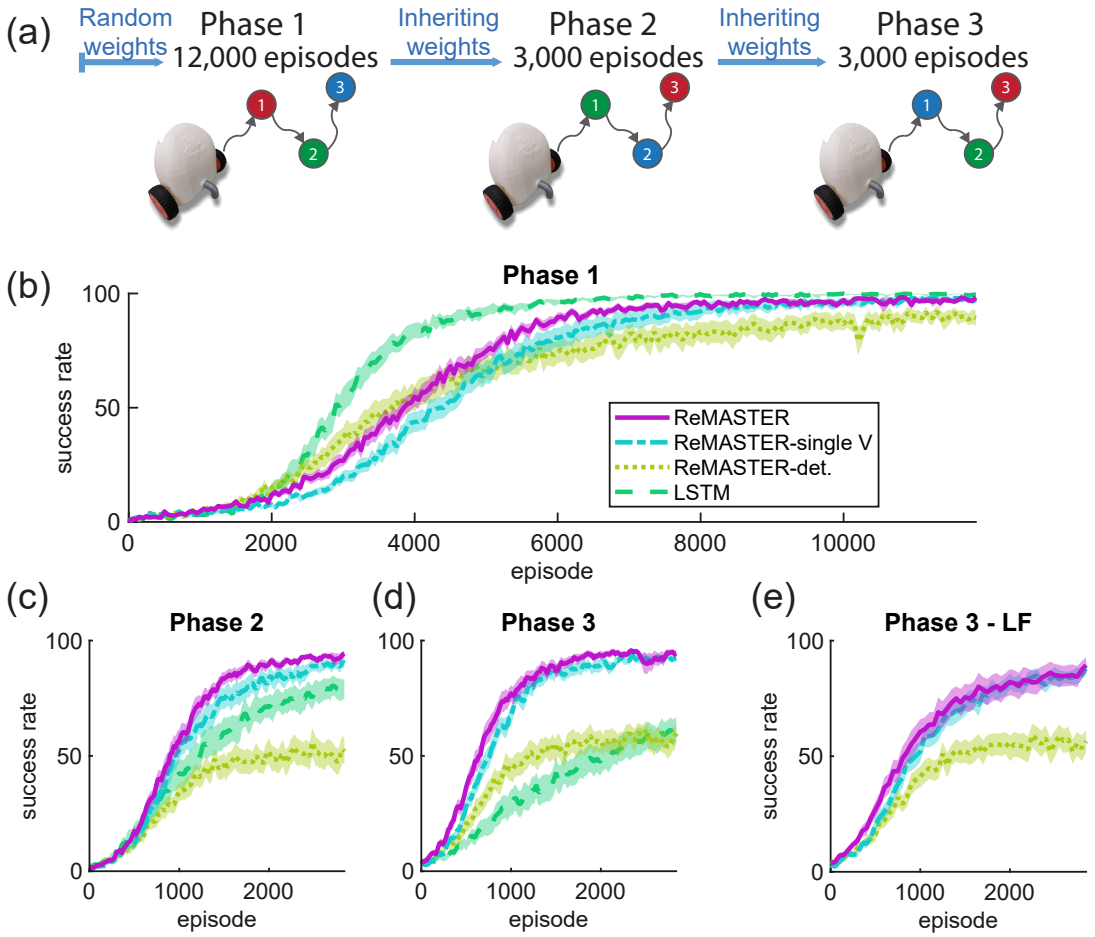


Figure 3.5: The consecutive relearning task: (**a**) Illustration of the task. (**b-d**) Performance curves for all phases, plotted in the same way as Fig. 3.3(b). ReMASTER-det. stands for the case in which all the neurons followed deterministic dynamics, and LSTM is the alternative using the same algorithm, but the network was a single-layer LSTM. **(e)** Performance curve of phase 3 with the lower-level synaptic weights frozen (Phase 3 - LF). In phases 2 and 3 (**c-e**), every model inherited its parameters from the previous phase.

zero. This is because the agent learned to follow the exact sequence of red-green-blue in phase 1, instead of randomly visiting them. Therefore, when the task switched to phase 2, where the desired sequence was green-blue-red, the agent still tried to reach red-green-blue by sequence. Since the agent did not learn to reach red after reaching blue in phase 1, the task mostly could not be completed at the beginning of phase 2. A similar reason applies to phase 3.

Second, ReMASTER significantly and consistently outperformed ReMASTER-det. in all three phases (Fig. 3.5(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 [61, 191], 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 to either the higher or lower level, and the results are shown in Chap. 3.4.13.

Third, ReMASTER also addressed consistent performance advantage over ReMASTER-single V (Fig. 3.5(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 [51, 215].

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. 3.5(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 the biological plausibility of our approach is arguable, this result may underlie a potential reason why we have many separated, timescale-distinct brain regions working for multiple levels of functions [154, 177, 238].

3.4.7 Learning new tasks with low-level weights frozen

The previous results (Fig. 3.5(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 the 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. 3.1) 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. 3.5(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 achieving sub-goals had developed in the low level, and memory for sequencing sub-goals had developed in the high level. This was facilitated by neuronal noise.

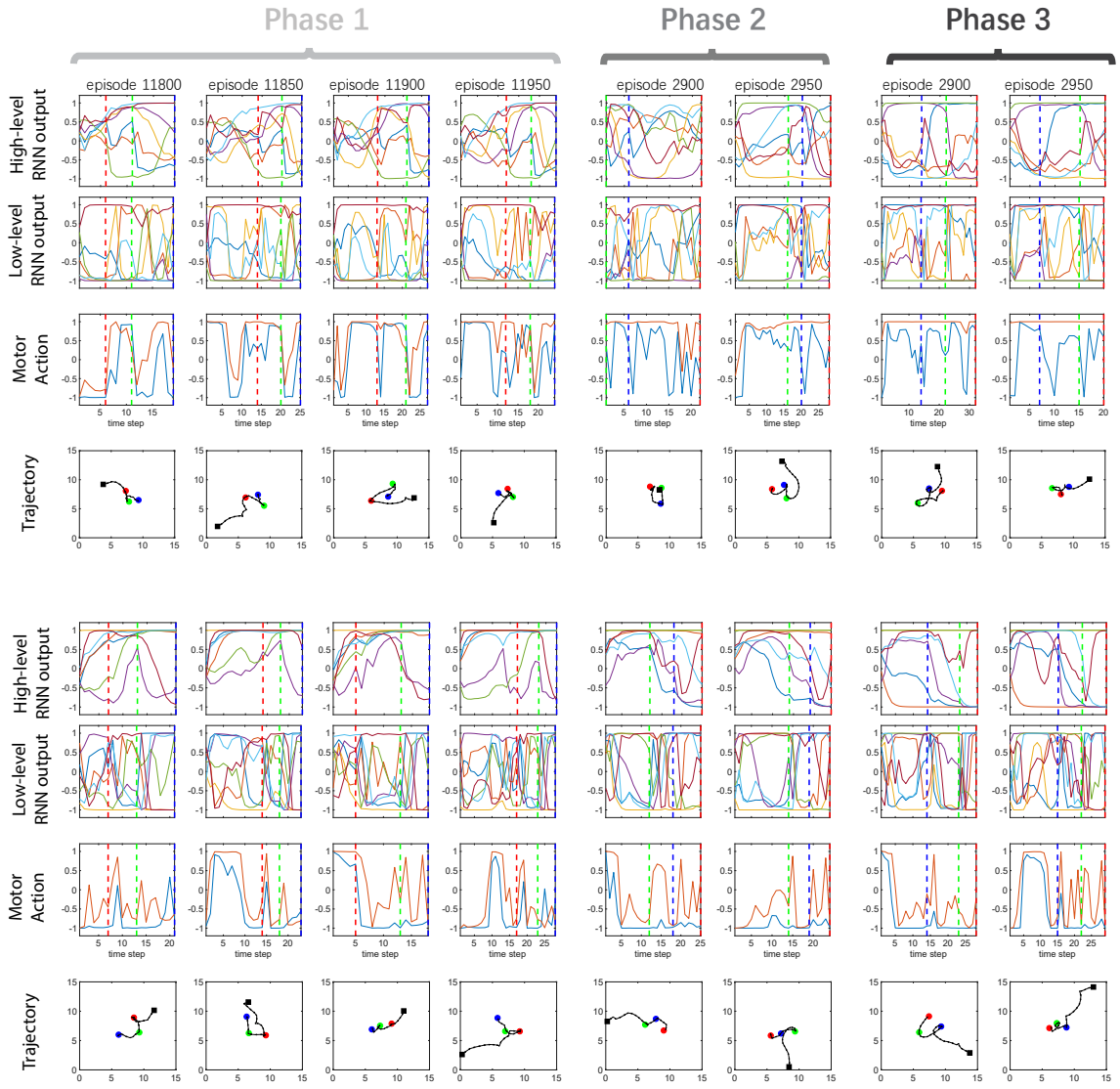


Figure 3.6: Example episodes showing the behavior of two well-trained ReMASTER agents in all 3 phases. Plotted in the same way as Fig. 3.4(a). For clarity, the first 7 neurons are plotted for both levels, with different colors indicating different neurons.

3.4.8 Consistency in representing sub-goals

To understand the underlying neural mechanisms for ReMASTER's promising performance in relearning phases (Fig. 3.5(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.

Table 3.2: Consistency of RNN outputs in representing sub-goals among the last 1,000 episodes in each phase. Data are Mean ± STD.

Network	Phase 1	Phase 2	Phase 3
ReMASTER (high level)	$\mathbf{0.95 \pm 0.02}$	$\mathbf{0.94 \pm 0.03}$	$\mathbf{0.95 \pm 0.02}$
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	$\mathbf{0.85 \pm 0.20}$	0.78 ± 0.20	0.54 ± 0.29

We measured consistency in representing sub-goals by cosine similarity of the temporal profile of the RNN outputs $\boldsymbol{c}^l$ across the last 1,000 episodes of each phase, for both the higher level and the lower one (Table 3.2). It can be seen that higher consistency mostly corresponds to a higher success rate for the three models in the consecutive relearning task (Fig. 3.5(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 Chap. 3.4.14 for more discussion). However, it is rather important that higher flexibility for re-organizing sub-goal representation was shown using ReMASTER agents.

3.4.9 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 the rodent motor cortex are thought to be important in action choices, while the primary motor cortex is considered responsible for details in action execution [150]. More interestingly, experimental studies have demonstrated that action primitives of animals can be altered by electrophysiological stimulation or optogenetic inactivation to certain upstream neurons [150, 236].

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 intermediate step of (i) from the 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

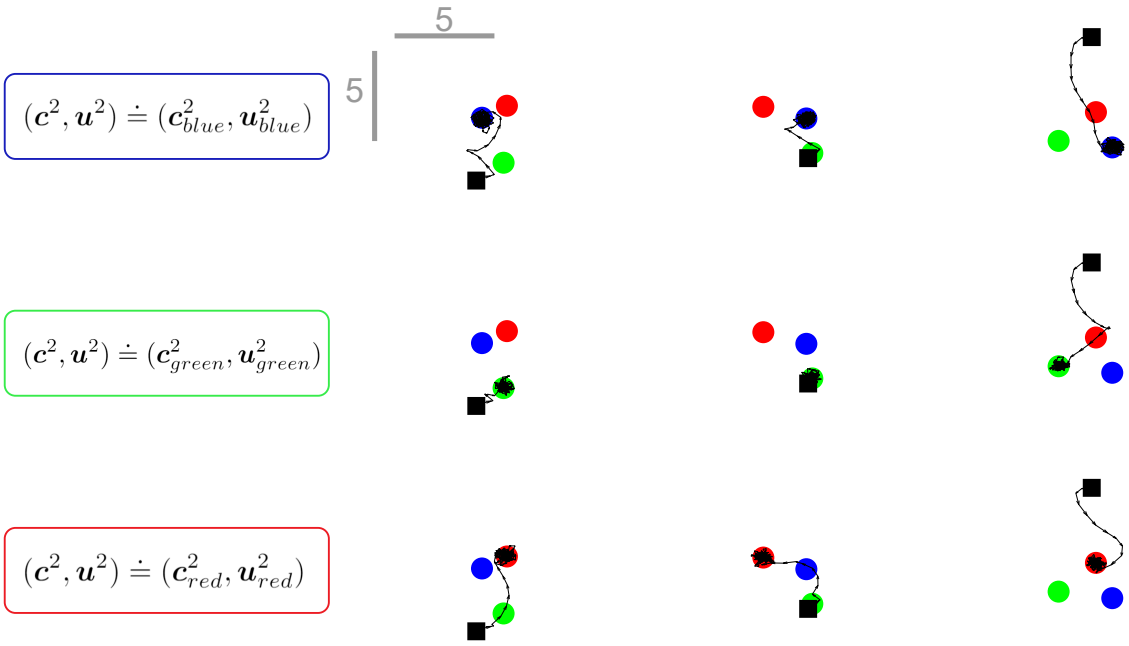


Figure 3.7: Manipulating agent behaviors by clamping high-level RNN states. All trajectories were from one agent, and each row used the same high-level RNN states. The black squares and the colored circles indicate the agent's initial and target positions, respectively. Each column used the same random seeds for generating initial and target positions.

trained agent to consistently follow an action primitive pursuing the corresponding sub-goal (Fig. 3.7). 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 the label for an action primitive. The continuous property of the RNN states enables the representation of an arbitrary number of sub-goals. We can readily find 3 meaningful action primitives corresponding to the 3 targets in our case.

3.4.10 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 $\tau^1(=2)$ and $\gamma^1(=0.92)$, corresponding to a fast dynamic, whereas the higher level was characterized by a slower timescale ($\tau^2 = 8, \gamma^2 = 0.98$). However, this "the-higher-the-slower" setting should also be validated.

For this purpose, different settings of τ^l and γ^l were examined in the consecutive relearning task. The simulation results (Fig. 3.8) 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.

3.4.11 Effect of hyperparameters

Many RL algorithms suffer from a proper choice of hyperparameters (such as the learning rate and the number of neurons in the network) for a satisfying performance. It is also important for us to ensure that our main results are robust to hyperparameters. For this purpose, we did a random search for hyperparameters (Fig. 3.9). 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. 3.9, although the overall performance was a little worse than that using tuned hyperparameters, our conclusions in Section. 3.4.6 did not vary. The LSTM alternative performed better in phase 1, but became worse in the later phases. Also, ReMASTER always outperformed ReMASTER-det. and ReMASTER-single V.

3.4.12 Comparing learning from scratch and relearning

We prepared a control task that is equal to phase 3 (also equivalent to phase 2 because of the symmetry of the three targets) except for a random initialization of synaptic weights at the beginning (Fig. 3.10, Bottom). It can be seen that agents with inherited weights largely outperformed agents in the control case that start from scratch, showing the meta-learning competency of RNNs.

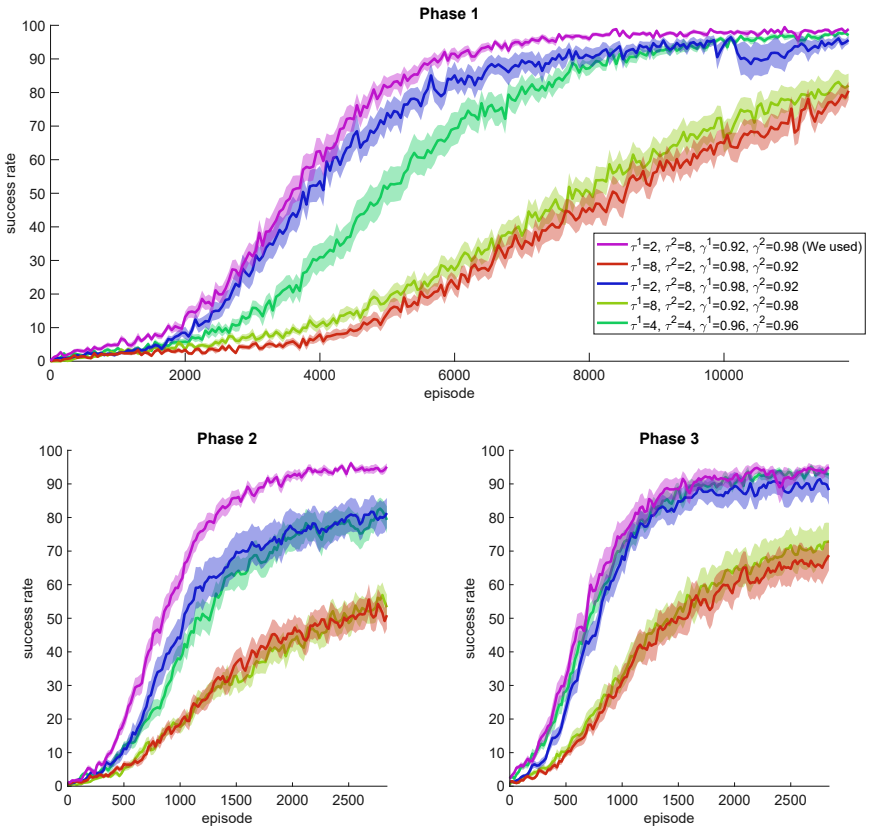


Figure 3.8: Performance comparison among different settings of τ^l and γ^l. Each result was obtained from 10 repeats.

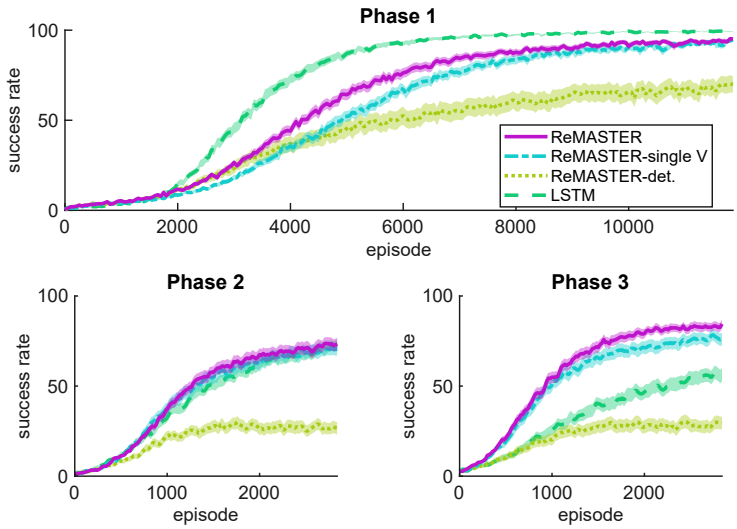


Figure 3.9: Performance in the consecutive relearning task, using a range of hyper-parameters.

3.4.13 Neuronal noise

We performed experiments to determine the proper value of σ_0, and found that $\sigma_0 = 0.2$ gave rise to better performance (Fig. 3.11). Thus we used $\sigma_0 = 0.2$.

We further conducted experiments to investigate the role of neuronal noise in either the higher level or the lower level. The results (Fig. 3.12) show that lack of neuronal noise in the higher level led 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 stochasticity in the higher level led to slightly worse performance in all 3 phases.

3.4.14 Development of internal representations

To see how the 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[5] of a randomly selected ReMASTER agent (Fig. 3.13, Left). The agent was trained to learn the consecutive relearning tasks in 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. 3.13) to see how the representation varies for the same task goal but at different learning stages.

Consistent with the result that learning was faster in later phases (Fig. 3.5(b,c)), internal representation also converged faster in relearning phases (Fig. 3.13, Left). In particular, when the agent returned to phase 1, it only took less than 1,000 episodes

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

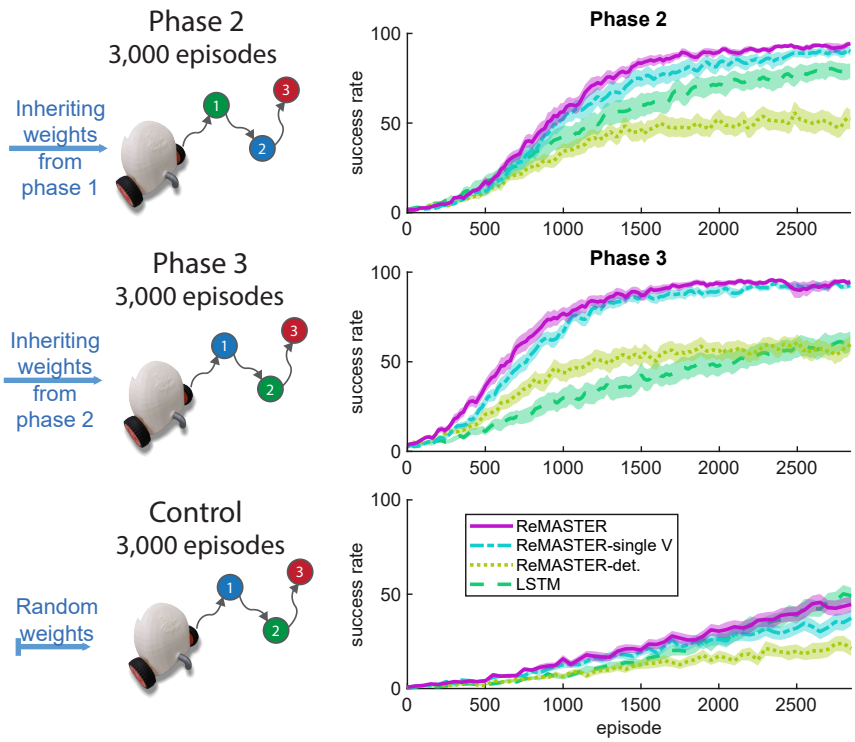


Figure 3.10: Performance comparison among phases 2, 3 and the control case.

to achieve a converged representation of sub-goals (The first 2 PCs after episode 1,000 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. 3.13, Right). This can also be demonstrated using the similarity measure (Chap. 3.4.8) between the RNN outputs in these two phases (averaged from the last 1,000 episodes in each phase), which is 0.37 $\pm$ 0.07 for the lower level and 0.33 $\pm$ 0.09 for the higher level, on 20 trials.

However, the first 2 PCs interestingly show that the representation at the end of phase 1 and phase 4 have similar structures, despite that the basis vectors are different (Fig. 3.13, 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 the last 1,000 episodes in each phase). The result showed a similarity of 0.93 $\pm$ 0.07 for the lower level and 0.97 $\pm$ 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 a sufficient amount of learning from scratch (phase 1) or by relearning, in which the internal representation was acquired much more quickly.

3.5 Scaling up to more challenging tasks

We have provided comprehensive empirical results of ReMASTER for the sequential reaching task in which the self-organized action hierarchy obtained by RL was shown to accelerate relearning in the recomposed relearning tasks. How well does ReMASTER

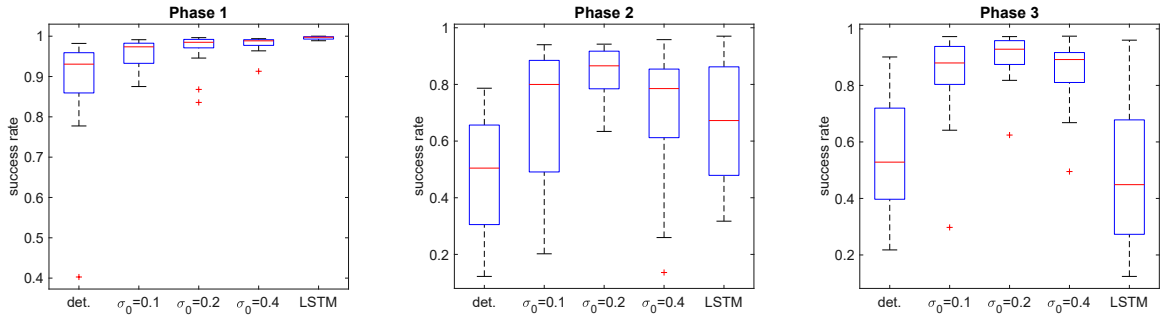


Figure 3.11: Final performances of ReMASTER for different scales of neuronal noise obtained from the last 1,000 episodes of each phase.

scale with more complex tasks? In this section, we extend ReMASTER to additional, more challenging tasks to examine its performance in more real-world-like tasks.

This section is arranged as follows: First, Chap. 3.5.1 will describe the additional tasks we used, which involve the challenge of higher-dimensional observation space (camera image) as well as higher-dimensional action space (7-joints robot arm). Then, Chap. 3.5.2 will explain the updated model implementation details of ReMASTER for the new tasks. Finally, the experimental results will be shown and discussed in Chap. 3.5.3.

3.5.1 Tasks

Vision-based sequential reaching task: To examine ReMASTER with a higher-dimensional observation space, we developed a task using the PyBullet simulator [37]. The task is similar to the sequential reaching task in Chap. 3.4 but with an RGBD camera as the sensor. We refer to this task as the *vision-based sequential reaching task*.

In this task, a two-wheeled robot can move in a square area surrounded by walls, as shown in Fig. 3.14(a). The task goal is to reach the 3 target positions, indicated by the 3 colored cylinders[6], according to the sequence of red-green-blue (Fig. 3.14(a)). When the robot reaches each target in the correct sequence, it will get a one-step reward (20, 50, and 100 for the 3 targets, respectively). When the robot has a collision with the wall, it will get a negative reward of -0.1. In each episode, the robot is initialized at the center of the task field (size 7×7). The 3 target positions are randomly set, while it is ensured that the distance between any two objects (the robot and targets) is at least 1.5. The robot is considered to reach a target if the horizontal distance between the centers of the robot and the target is less than 1. An episode terminates when the robot correctly reaches the 3 targets, or a time limit of 100 steps is reached.

As in the original sequential target reaching task, the velocities of the two wheels are set as the action, which is a two-dimensional continuous vector. However, the observation changes from the previous depth and angle sensors to a 360° RGBD camera installed on the robot. In particular, the observation in this task has a resolution of

[6]For simplicity, the cylinders have no physical collision with the robot, i.e., the robot can pass through them.

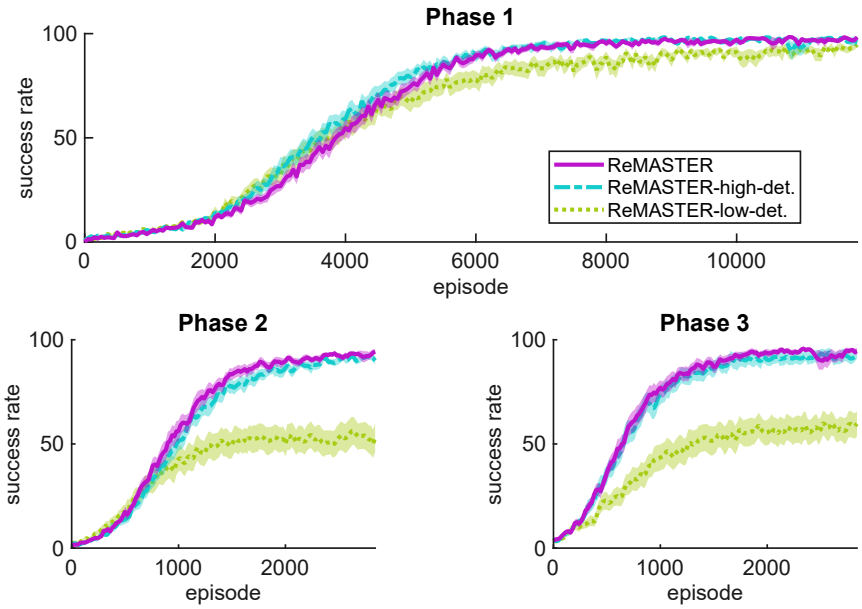


Figure 3.12: Success rate in all 3 phases. ReMASTER is compared to ReMASTER-high-det. and ReMASTER-low-det., in which the higher-level or the lower-level neurons are deterministic.

$64 \times 16 \times 4$, where the last dimension corresponds to RGBD channels. An example of the robot's RGB vision is shown at the bottom of Fig. 3.14(a).

Robot arm sequential touching task: We also tested ReMASTER using another kind of robotic control task with a higher-dimensional action space. In this so-called *robot arm sequential touching task* (Fig. 3.14(b)), a 7-joints robot arm is trained to touch 3 target positions sequentially. The task is modified from the *reach* task of the panda-gym environments [74]. The robot arm is considered to reach a target position when its end-effector overlaps with the target. The end-effector is the tip of the gripper, where the gripper is always closed (Fig. 3.14(b)).

In each episode, the robot arm is initialized as shown in Fig. 3.14(b). 3 target positions are randomly sampled in a cubic space in front of the robot arm, while it is ensured that the 3 targets are not too close to each other. One-step rewards (20, 50, and 100 for the 3 targets, respectively) are given at each target position if the end-effector touches it in the correct sequence, which is the same as the previous tasks. For consistency, we also colored the 3 targets red, green, and blue. There are no other rewards or punishments.

The observation is the concatenation of joint angles, the x-y-z position of the end-effector, and the x-y-z positions of the red, green, and blue targets. Note that unlike a goal-directed RL setting where the goal state is clearly specified [8], the agent does not know which part of the observation corresponds to the current target.

As for the action setting, the panda-gym environments provide two control modes: end-effector displacement control (3-dimensional continuous action) and joint velocities control (7-dimensional continuous action). We conducted experiments using both control modes.

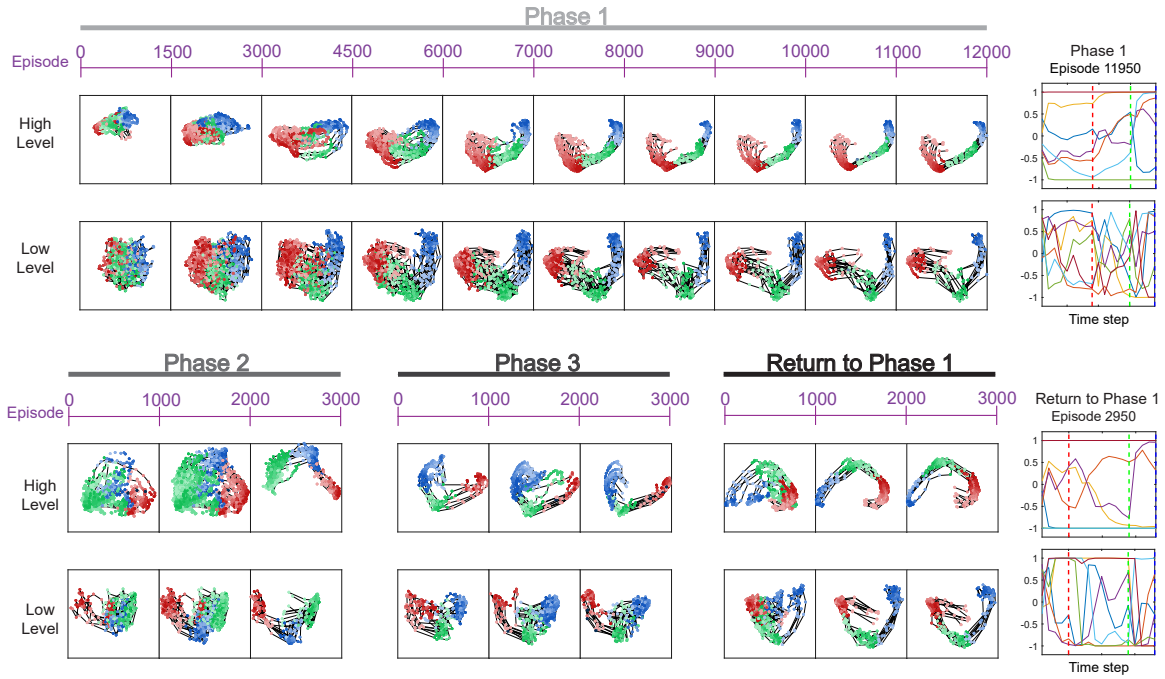


Figure 3.13: Development of internal representation for sub-goals. Left: The first two PCs of internal representations of a ReMASTER agent using PCA for each phase separately, plotted in the same way as Fig. 3.4(b). Right: Profiles of RNN outputs of the higher level (Top) and the lower level (Bottom), plotted in the same way as Fig. 3.4(a).

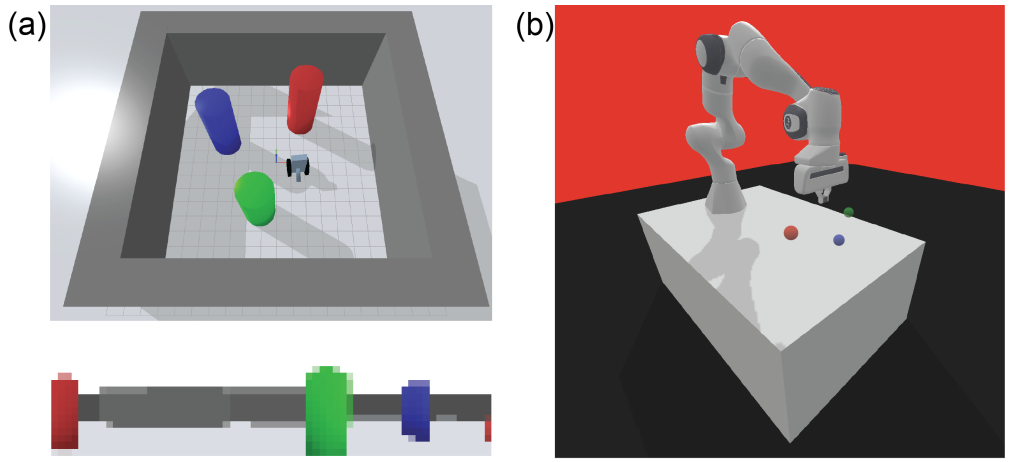


Figure 3.14: Rendering of the environments used in Chap. 3.5. (**a**) Top: the vision-based sequential reaching task. Bottom: an example of the RGB image from the 360° camera installed on the robot. (**b**) The robot arm sequential touching task.

Relearning tasks: Similar to the relearning phases in Chap. 3.4.6, we changed the required target sequence of the vision-based sequential reaching task and the robot arm sequential touching task to investigate the agent's transfer learning capacity. We conducted 3-phases experiments as in the consecutive relearning task (Chap. 3.4.6), where the network weights and biases were carried from one phase to the next, and the replay buffer was initialized at the beginning of each phase. The required sequence was red-green-blue in phase 1, green-blue-red in phase 2, and blue-green-red in phase 3, while other conditions kept the same in all phases.

3.5.2 ReMASTER implementation for the additional tasks

To enable ReMASTER to solve the more challenging task, there were some changes from the implementation in 3.4. However, the two most essential ideas of ReMASTER, i.e., the intrinsic hierarchy of timescales in the RNN and the neuronal noise, remained intact. Correspondingly, we stuck to the MTSRNN architecture as in 3.4. This section details the updated implementation of ReMASTER.

RL algorithm: To apply ReMASTER to the more challenging tasks, we adopted the recently developed off-policy RL algorithms for continuous control to replace the original policy evaluation and policy improvement algorithms (Chap. 3.4.2) since ReMASTER is flexible to the choice of RL algorithm. In particular, we tested ReMASTER with two popular off-policy algorithms known as twin delayed deep deterministic policy gradient (TD3, proposed by Fujimoto et al. [70], see Chap. 2.1.11) and soft actor-critic (SAC, introduced by Haarnoja et al. [84], see Chap. 2.1.10).

Network: As we are dealing with the tasks with higher-dimensional observation and actions spaces, we increased the number of neurons in the network. For ReMASTER (and its ablations), the lower and higher level had 256 and 128 neurons, respectively.

Since TD3 and SAC learn Q-functions instead of the state value function V, each layer's RNN output $\boldsymbol{c}^l$ was concatenated with the action as the input of the Q-function in experience replay. Note that TD3 and SAC uses two Q-networks to mitigate the overestimation of the Q-value [70, 84]. Thus, in the new implementation of ReMASTER, we had two Q-networks for each RNN layer with the corresponding discount factor γ^l, where each Q-network was an MLP with 1 hidden layer.

For TD3, the policy network was a one-layer MLP mapping the lower-level RNN output $\boldsymbol{c}^1$ to the non-noisy action (using a hyperbolic tangent function to bound its range). For SAC, the policy network was also a one-layer MLP mapping the lower-level RNN output to $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. Then the action was sampled as $\boldsymbol{a} = \tanh(\boldsymbol{u})$, where $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$.

For the robot arm sequential touching tasks whose observations were vectors, the raw observation was directly passed to the lower-level RNN as the input at each time step. For the vision-based reaching task, we used a 2-D CNN to process the 2-D pixels observation before inputting it to the lower-level RNN. The CNN is detailed in Table 3.3. All the hidden layers were of width 256 and with hyperbolic tangent nonlinearity except those in the CNN and RNN. All the networks were trained as a whole.

layer	module	channels	stride	kernel size	activation
1	Conv2D	8	2 by 2	4 by 4	ReLU
2	Conv2D	16	2 by 2	4 by 4	ReLU
3	Conv2D	16	2 by 2	4 by 4	ReLU
4	Conv2D	64	2 by 2	4 by 2	ReLU
5	Conv2D	256	4 by 1	4 by 1	ReLU
6	Flatten	256			

Table 3.3: Details of the CNN layers used for the vision-based sequential reaching task. The input resolution is 64 by 16.

Alternatives agents: As in Chap. 3.4, we also conducted experiments using alternative agents of ReMASTER. The first one was ReMASTER - single V, which was the same as ReMASTER except that the higher-level value function was not learned. The second was ReMASTER - det., which was the ablation of ReMASTER without neuronal noise. The third used a 1-layer LSTM (without neuronal noise) and learned the value function with an intermediate discount factor, which can be considered as the ablation of the intrinsic hierarchy of timescales and neuronal noise. The number of neurons in the 1-layer LSTM is 215. In addition, we tested a 2-layers LSTM agent (2L-LSTM), which was the same as ReMASTER - det. except that the 2-levels MTSRNN was replaced with 2 layers of LSTM. The numbers of neurons in the 2-layer LSTM were 160 and 80 for the lower and higher levels, respectively. The numbers of neurons in the LSTM and the 2L-LSTM were decided so that the total amount of model parameters were similar to that of ReMASTER (around 0.76 million for the vision-based sequential reaching task).

Initial exploration: We used another trick to improve the performance of all the models. For a given number of steps at the beginning of the task (25,000 steps in our implementation), the agent executes actions sampling from a uniform random distribution covering the whole action space (bounded continuous actions for the tasks we used). This trick is also used in some well-established RL implementations, such as the OpenAI Spinning Up [2]. The initial exploration did not apply to the relearning cases.

Hyperparameters: The hyperparameters are detailed in Table 3.4. For those unmentioned hyperparameters specific to TD3 and SAC, we followed the original papers [70, 84].

3.5.3 Experimental results

We show the results of the vision-based sequential reaching task (Fig. 3.15a,b) and the robot arm sequential touching tasks (Fig. 3.16a,b) using ReMASTER and the alternative models. The agents performed lifelong RL in three consecutive learning phases like in the consecutive relearning task (Chap. 3.4.6). We conclude the experimental results about sample efficiency as follows.

First, ReMASTER and ReMASTER - single V were the most well-performing models overall (Fig. 3.15a,b and 3.16a,b), except that ReMASTER - single V struggled with

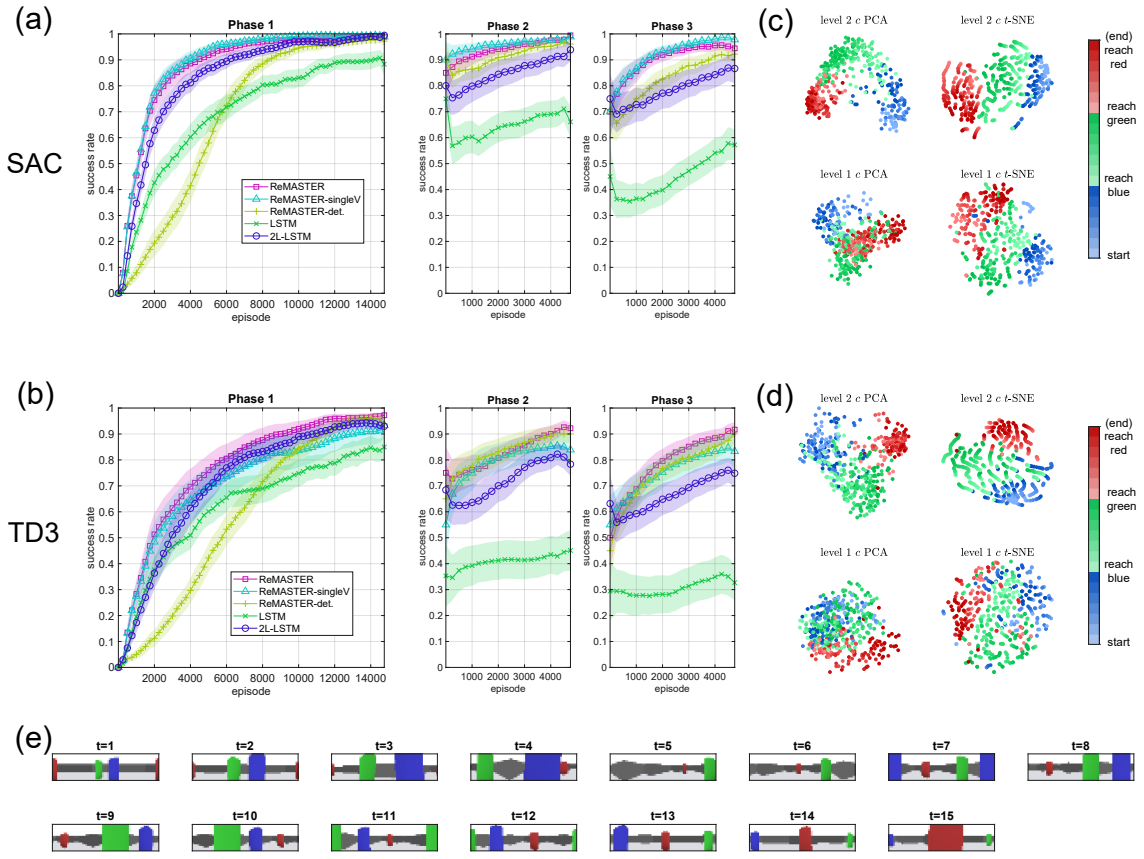


Figure 3.15: (**a,b**) Results of the vision-based sequential reaching task, as well as the relearning phases, where the RL algorithm used was (a) SAC (b) TD3. The performance is indicated by success rate, where success is defined as task accomplishment within 50 steps, the same as in Fig. 3. Each experiment was run using 20 random seeds. (**c,d**) Examples of the learned internal representation of ReMASTER in phase 3, where the RNN outputs $\boldsymbol{c}^1$ and $\boldsymbol{c}^2$ were dimension-reduced by PCA and t-distributed stochastic neighbor embedding (t-SNE) [230]. The data for PCA/t-SNE were obtained from the successful trials in the last 1,000 episodes. (**e**) The observations in an example episode of a ReMASTER agent at the end of phase 3, where the agent reached the blue, green, and red targets sequentially.

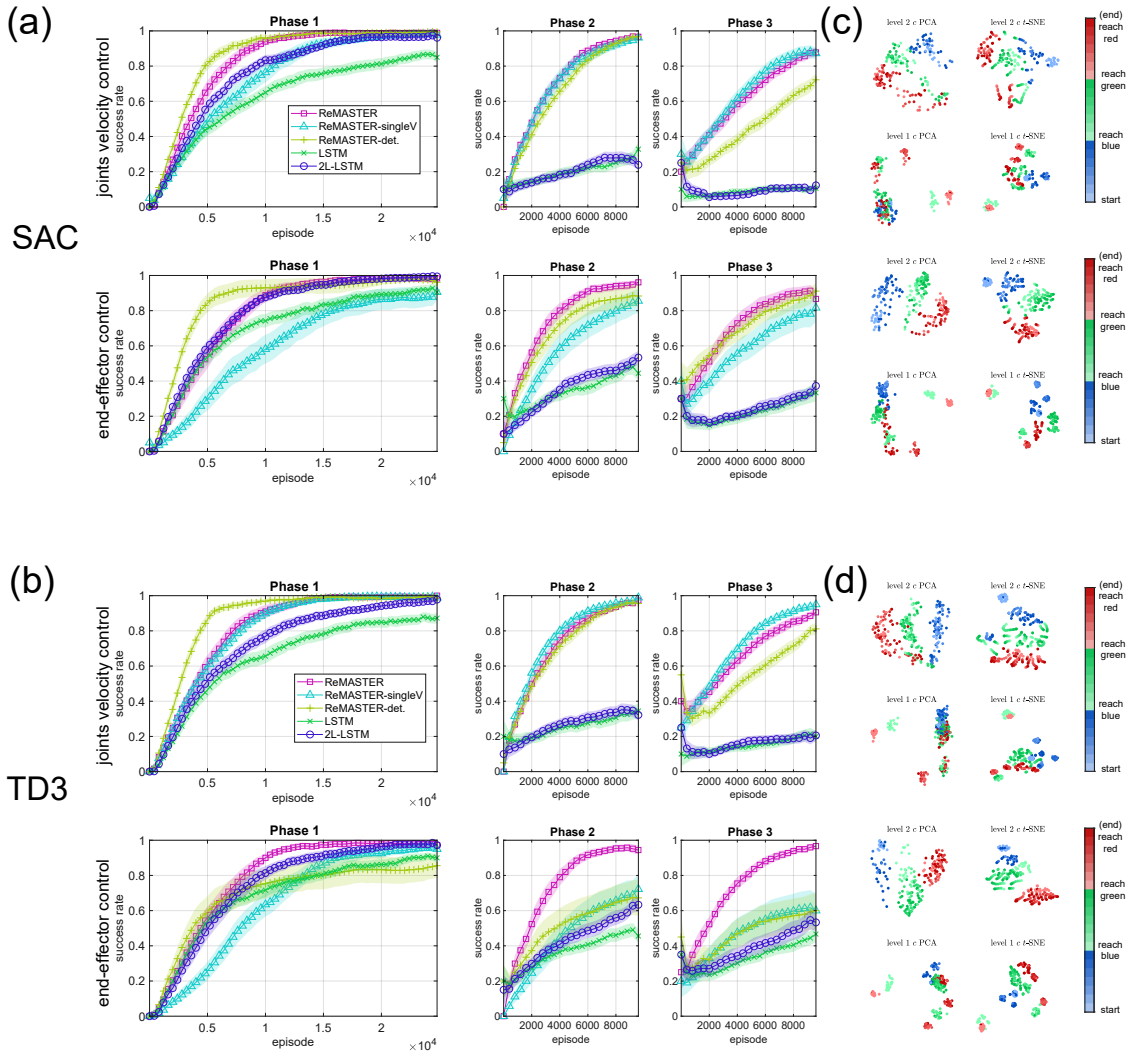


Figure 3.16: (**a,b**) Results of the robot arm sequential touching tasks (including the two cases of joints velocity control and end-effector control), as well as the relearning phases, where the RL algorithm used was (a) SAC (b) TD3. (**c,d**) Examples of the learned internal representation of ReMASTER in phase 3. The figures were plotted in the same way as those in Fig. 3.15.

Table 3.4: Hyperparameters of the updated implementations of ReMASTER.

Hyperparameter	Description	Value	comment
γ^1	Low-level discount factor	0.92	same as previous
γ^2	High-level discount factor	0.98	same as previous
τ^1	Low-level RNN timescale	2	same as previous
τ^2	High-level RNN timescale	8	same as previous
N^1	Number of neurons in the lower level	256	larger than previous
N^2	Number of neurons in the higher level	128	larger than previous
buffer_size	Number of steps recorded in memory	1e6	larger than previous
σ	Scale of neuronal noise	0.1	fixed (previously annealed)
e	Scale of motor noise	0.1	fixed (previously annealed)
σ_a^{TD3}	Scale of target policy noise	0.1	for TD3 [70] only
n_update	Number of steps per update	2	same as previous
lr_critic	Learning rate of critic	3e-4	follow SAC&TD3 [70, 83]
lr_actor	Learning rate of actor	3e-4	follow SAC&TD3 [70, 83]
α	Decay of the RMSProp optimizers	0.99	same as previous
L	Sequence length for truncated BPTT	64	larger than previous
batch_size	Number of training sequences	8	smaller than previous

the robot arm sequential touching task with end-effector control (Fig. 3.16b). ReMASTER performed the best or comparable to the best in all the relearning cases (phases 2, 3). Also, ReMASTER and ReMASTER - single V demonstrated their transfer learning capacity in the relearning phases, especially in the vision-based sequential reaching task (Fig. 3.15a,b). For the robot arm sequential touching tasks, ReMASTER also achieved better sample efficiency in the relearning phases than from scratch (Fig. 3.16a,b).

Second, ReMASTER - det. performed comparably to ReMASTER when learning from scratch (phase 1, Fig. 3.15 and 3.16a,b). However, the lack of neuronal noise in the relearning cases led to consistently worse performance than ReMASTER, if not similar, in all the tasks. Such results indicate the benefit of stochastic neural activities for transfer learning in the being tested tasks.

Third, LSTM and 2L-LSTM showed unsatisfactory performance compared to ReMASTER in the relearning phases of all the tasks, although their performance in phase 1 was comparable to ReMASTER (Fig. 3.15a,b and 3.16a,b). Since the LSTM and 2L-LSTM models did not have an intrinsic hierarchy of timescales in the network, they probably developed internal representations that were specifically good for phase 1 but inflexible for skill transfer.

Now we switch our attention to the internal representation. Did the ReMASTER self-develop hierarchical representations in the RNN states? We visualized the learned representation of the RNN outputs of both levels of an example ReMASTER agent (Fig. 3.15c,d and 3.16c,d). For the vision-based sequential reaching task, sub-goals were more recognizable in the RNN outputs of the higher level than the lower level (Fig. 3.15c,d). Such difference was more evident for the robot arm sequential reaching tasks (Fig. 3.16c,d). For a numerical evaluation, We calculated the consistency in representing sub-goals for each layer of each model like in Chap. 3.4.8. The results are shown in Table. 3.5. The highest consistency in representing sub-goals was from the high-level RNN of ReMASTER - det. in phase 1, and from the high-level RNN of

ReMASTER in the relearning phases.

Table 3.5: Consistency of RNN outputs in representing sub-goals for the vision-based sequential reaching task and the robot arm sequential touching tasks. The data were obtained from the last 1,000 episodes of each phase for each model regardless of the RL algorithm used. Data are Mean ± STD.

Task	Network	Phase 1	Phase 2	Phase 3
Vision-based sequential reaching	ReMASTER (high level)	0.61 ± 0.10	**0.61 ± 0.11**	**0.62 ± 0.12**
	ReMASTER (low level)	0.45 ± 0.07	0.41 ± 0.07	0.39 ± 0.07
	ReMASTER - single V (high level)	0.47 ± 0.09	0.46 ± 0.11	0.48 ± 0.11
	ReMASTER - single V (low level)	0.46 ± 0.06	0.42 ± 0.06	0.40 ± 0.06
	ReMASTER - det. (high level)	**0.65 ± 0.08**	0.56 ± 0.09	0.55 ± 0.10
	ReMASTER - det. (low level)	0.46 ± 0.06	0.34 ± 0.06	0.32 ± 0.07
	2L-LSTM (high level)	0.54 ± 0.06	0.58 ± 0.06	**0.62 ± 0.05**
	2L-LSTM (low level)	0.36 ± 0.04	0.32 ± 0.03	0.36 ± 0.03
	LSTM	0.31 ± 0.04	0.35 ± 0.04	0.35 ± 0.03
Robot arm sequential touching (joints control)	ReMASTER (high level)	0.61 ± 0.04	**0.54 ± 0.05**	**0.48 ± 0.08**
	ReMASTER (low level)	0.40 ± 0.04	0.37 ± 0.04	0.32 ± 0.06
	ReMASTER - single V (high level)	0.44 ± 0.08	0.43 ± 0.07	0.38 ± 0.07
	ReMASTER - single V (low level)	0.39 ± 0.06	0.40 ± 0.10	0.34 ± 0.08
	ReMASTER - det. (high level)	**0.66 ± 0.04**	0.49 ± 0.06	0.32 ± 0.07
	ReMASTER - det. (low level)	0.42 ± 0.05	0.37 ± 0.04	0.28 ± 0.04
	2L-LSTM (high level)	0.45 ± 0.08	0.19 ± 0.06	0.17 ± 0.08
	2L-LSTM (low level)	0.49 ± 0.06	0.20 ± 0.06	0.21 ± 0.06
	LSTM	0.23 ± 0.07	0.16 ± 0.04	0.17 ± 0.08
Robot arm sequential touching (end-effector control)	ReMASTER (high level)	**0.60 ± 0.05**	**0.64 ± 0.07**	**0.61 ± 0.06**
	ReMASTER (low level)	0.42 ± 0.05	0.47 ± 0.08	0.43 ± 0.06
	ReMASTER - single V (high level)	0.34 ± 0.12	0.42 ± 0.13	0.44 ± 0.10
	ReMASTER - single V (low level)	0.47 ± 0.07	0.44 ± 0.14	0.43 ± 0.10
	ReMASTER - det. (high level)	**0.61 ± 0.12**	0.59 ± 0.10	0.49 ± 0.12
	ReMASTER - det. (low level)	0.49 ± 0.10	0.43 ± 0.10	0.35 ± 0.09
	2L-LSTM (high level)	0.45 ± 0.08	0.23 ± 0.10	0.24 ± 0.12
	2L-LSTM (low level)	0.40 ± 0.07	0.22 ± 0.10	0.26 ± 0.10
	LSTM	0.22 ± 0.09	0.15 ± 0.03	0.18 ± 0.04

The results discussed above were consistent with those for the simpler sequential target reaching task on which we focused in Chap. 3.4. Thus, we showed that ReMASTER is scalable to more real-world tasks where several sub-goals need to be accomplished sequentially. Our experiments suggest that the model's intrinsic hierarchy of timescales and stochasticity were crucial for self-organization of action hierarchy and relearning in new tasks with recomposed sub-goals.

3.6 Summary

In this work, we performed empirical investigations on how sequential compositional tasks can be solved by autonomously developing sub-goal structures 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, inspired by neuroscientific findings [101, 154, 156, 177, 199]. The other is stochasticity introduced in neural units in all RNN layers, also inspired by the corresponding biological facts [14, 15, 160].

While a great number of HRL approaches already exist (Chap. 3.2), ReMASTER is not incremental to any of them but rather a distinct framework for HRL. As we discussed in Chap. 3.2, the HRL approaches can be divided into two categories: temporally extended (option framework) [10, 11, 39, 45, 60, 63, 121, 130, 132, 165, 174, 210, 221, 246, 251] and latent-sub-goal HRL [3, 8, 53, 79, 129, 155, 180, 189, 190, 232, 247].

ReMASTER differs from the temporally extended HRL approaches in terms that the higher-level action of ReMASTER is a continuous vector, or more specifically, the internal states of the higher-level RNN, which change all the time. However, temporally extended control can be achieved by clamping the high-level RNN states of ReMASTER to fixed values, as shown in Chap. 3.4.9.

While ReMASTER falls into the latent-sub-goal HRL category since the high-level RNN states can be considered the latent sub-goal, ReMASTER is unique. All the previous latent-sub-goal HRL methods, to our knowledge, took advantage of pseudo rewards or objective functions for action abstraction, in addition to the original RL objective (policy evaluation and policy improvement based on the original reward function and action space of the task). ReMASTER fundamentally differs from the other latent-sub-goal HRL methods in that ReMASTER does not rely on a particular pseudo reward or objective function for action abstraction; instead, ReMASTER employs a multiple-timescale RNN architecture which implicitly contributes to the self-organization of action hierarchy via RL. In other words, ReMASTER's novelty is mainly about the neural network architecture, which is orthogonal to the prior methods' contributions that are mostly algorithmic. It is worth mentioning that our work does not suggest that ReMASTER is superior to other HRL methods; instead, it should be interesting to investigate how to incorporate ReMASTER with existing HRL algorithms in the future.

To empirically demonstrate the effectiveness of ReMASTER, we first conducted a comprehensive case study using the sequential target-reaching task in Chap. 3.4. Simulation results showed that action hierarchy emerged by developing an adequate internal neuronal representation at multiple levels. We presented several pieces of evidence 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, whereas a set of skills for detailed sensory-motor control for achieving each sub-goal acquired in the lower level. Furthermore, the compositionality developed via RL enabled efficient relearning in adaptation to changed task goals that involved the re-composition of previously learned sub-goals. This re-composition capability was further enhanced with the introduction of neuronal noise in addition to motor noise. Such adaptation became possible because the development of hierarchical control using multiple levels allowed enough flexibility for the re-composition of previously learned control skills. The results of the sequential target reaching and consecutive relearning tasks are proof of concept that an interpretable and composable action hierarchy can emerge via RL without any objective function for action abstraction.

We then scaled ReMASTER up by employing more recent off-policy RL algorithms

[70, 83] and tested it in the more challenging tasks (Chap. 3.5). The experiment results demonstrated the advantages of ReMASTER. In particular, ReMASTER significantly outperformed the LSTM agents on sample efficiency in the relearning phases, consistent with the results in Chap. 3.4.

Our results altogether highlighted the potential of brain-inspired network structures for deep HRL. Also, our results provided potential insights into neuroscience research, which will be discussed in the next section.

3.7 Neuroscience Insights

3.7.1 Multiple timescales

How important is the intrinsic timescale hierarchy in the brain? While Murray et al. [154] found ascending (the-higher-the-slower) intrinsic timescales (of fluctuations in spiking activity) in the cortical areas of primates, this question remains difficult to answer in neuroscience due to the difficulty of conducting control experiments by changing the timescales of brain regions.

Computational models are convenient for changing the intrinsic timescales. A classic study by Yamashita and Tani [248] used a multiple-timescale RNN to perform supervised learning for robotic control. They tested different timescale settings for the 2-levels RNN and found that ascending timescales were beneficial for the learning performance. When it comes to self-exploratory learning (RL), our simulation results (Chap. 3.4.10) also suggested that ascending timescales setting in our model should be adopted for better performance. However, even with computational models, it is hard to test all the possible combinations of τ^1 and τ^2, not to say that we can stack more levels to the MTSRNN.

Interestingly, Chaudhuri et al. [25] suggested that a hierarchy of timescales can naturally emerge in a large-scale dynamical model of the macaque neocortex. We conjecture that such a hierarchy in the cortical areas is essential for the survival of animals, which implies that hierarchical property has probably been evolved by natural selection [168]. However, neuroscience and machine learning have not thoroughly investigated how the timescale hierarchy contributes to detailed tasks. Our study focused on the hierarchical control aspect and provided computational results for the control tasks with sequential sub-goals. Nonetheless, more future studies remain to be conducted toward a comprehensive understanding of the role of multiple timescales in neural networks.

3.7.2 Discount factor

While most RL studies consider solving an MDP or POMDP problem with a single discount factor γ, a few studies have investigated how multiple discount factors for a single task can facilitate RL [122, 172, 173]. Kurth-Nelson & Redish [122] proposes to employ multiple "micro-agents", each with a distinct γ for exponential discounting (as in normal RL), to constitute an "overall-agent" with hyperbolic discounting, consistent with the behavioral experiments. The γ-Ensemble method proposed by Reinke et al.

[172, 173] learns multiple Q-functions with different discount factors and uses the Q-function with the γ that maximizes the average reward to make decisions.

Our work provides a novel thought on how multiple discount factors can benefit RL. In ReMASTER, policy evaluation (learning value function) is conducted in multiple levels with different discount factors, where the higher level learns the value functions[7] with a higher γ (longer horizon). Note that the policy is learned using the lower-level value functions. Thus, the learning of the higher-level value functions can be considered as an auxiliary learning objective. Our ablation studies, which compared ReMASTER with ReMASTER-single V (higher-level value functions not learned), showed that such an auxiliary learning objective is helpful in some HRL tasks, perhaps by helping the higher-level to better representing sub-goals. This result provides further empirical evidence, in addition to existing studies [122, 172, 173], of the advantage of learning value functions with different discount factors in mammal brains, which was experimentally confirmed [51, 215].

An interesting future direction is to consider γ to be adaptive. This is related to the neurotransmitter serotonin (5-HT), which has been shown to play important roles in decision-making tasks [48], such as modulating risk-taking [175], learning rate [23, 142], trade-off between exploitation and exploration [23] etc. In particular, inhibiting or activating serotonin releasing modulates the animal or human's "patience" in a decision-making task, which corresponds to the discount factor in RL [147, 186, 187]. However, relatively few deep RL studies consider the discount factor(s) to be variable since it breaks the definition of an MDP or POMDP that has a single, fixed γ [16]. Computationally understanding the effect of variable γ remains an under-explored problem, although it is beyond the scope of this thesis.

3.7.3 Neuronal noise

Our results have shown the benefits of the neuronal noise in ReMASTER, which is implemented in a straightforward way—adding a Gaussian white noise to the hidden states (Eq. 3.2). Without particular treatment of the neuronal noise (e.g., Bayesian objective functions), the neuronal noise in ReMASTER improves both the sample efficiency and the consistency in representing sub-goals. This is a potential reason to account for the existence of stochastic neural activities in the brain (in particular, in the cerebral cortex) [14, 15, 91], that is, encouraging exploration of the high-level, abstracted behaviors (and probably also thinking and reasoning).

Since humans and animals need to learn to perform extensive and diverse tasks in their life (except for the innate abilities to perform the most important tasks, such as breathing), a general neural mechanism for action abstraction and exploratory behavior should be very helpful. Our results suggest that simply having noise in the neuronal dynamics can be one of the mechanisms. Moreover, our experiments of ReMASTER in various tasks have shown that in most cases, the neuronal noise enhanced performance, in comparison to ReMASTER - det.

A more advanced way of considering the neuronal noise is the Bayesian approach

[7] Here we refer to both the state value function V and the state-action value function Q as "value functions"

to brain function [47, 49]. Different from the relative naïve way of neuronal noise injection in ReMASTER, the Bayesian brain approach considers the neural encoding as a probabilistic variable and takes advantage of the mathematical concepts from the probabilistic and statistical theory such as Bayes' theorem. In deep RL, for example, Han et al. [90] used a Bayesian variable to encode the environmental state and train the model with information-theoretical objective functions such as log-likelihood and KL-divergence (by contrast, ReMASTER minimizes the mean square error between the target and output). A potential future direction to incorporate Bayesian modeling approaches into ReMASTER, in which the scale of neuronal noise is adaptive and learnable, as in the work of the next chapter. It will be interesting to investigate how the model trades off between the accuracy (lower variance of neuronal noise) and exploration (higher variance of neuronal noise) [7, 226].

Chapter 4

Variational RNN for RL in Partially Observable Environments

Note: *This section reused the thesis author's publication [88] (the other two authors of the paper are the thesis author's supervisor and co-supervisor) with modification and re-organization to fit the thesis.*

- Dongqi Han, Kenji Doya, and Jun Tani. "Variational recurrent models for solving partially observable control tasks." *International Conference on Learning Representations (ICLR)*, 2020.

4.1 Background

In the last chapter, we have discussed autonomously learning an action hierarchy in a model-free setting. We focused on sub-goal composition, while each sub-goal (reaching one target) was relatively simple to learn. In the real world, learning to control usually yields more challenges, such as high dimensional state/action spaces and partially observable environments [9].

In particular, it remains a variety of difficulties to solve partially observable (PO) tasks (those can be defined using POMDP, see Chapter 2.1.3). In PO tasks, the information from raw observation of the current timestep is often insufficient for optimal decision-making. It is usually assumed that the entire history of observations (i.e., the memory of observations) can be used for extracting underlying information about the environment, which contributes to a better policy or even optimal policy [88, 181].

An obvious problem is how to extract useful information for RL from historic observations? In some cases, it is easy to hard-code. For instance, in a snooker game, the striker should pot a colored ball after potting a red ball and vice versa. However, in more general cases, such a simple encoding cannot be easily given in prior. One straightforward solution is to use the whole history of observations as input to a neural network and use regular model-free reinforcement learning algorithms to train the network [125, 144]. Unfortunately, this solution has poor scalability: when the state space or task horizon is large, the dimension of neural network input will correspondingly explode, which brings considerable difficulty to training the network.

Such a problem can be overcome using RNNs as the function approximators, which provide flexibility for a variable length of observation sequence [102, 103, 110, 181, 183]. With the usage of RNNs, an arbitrary model-free RL algorithm should be able to solve PO tasks in theory, provided that the RNN used has sufficient expressive power for extracting useful information from a history of observations. However, meanwhile, the RNN has to learn to estimate value function and policy function, usually using bootstrapping strategy (i.e., the gradient of value function depends on the current estimation of value function [208]). Tackling these two problems poses difficulty to stable and efficient train the network since RNNs are known to be harder to tame than FNNs [166].

A substituted approach is to make use of a *belief state*, extracted from a sequence of state transitions [82, 107, 125], which usually requires learning a world model. The belief state is a variable estimated by the agent. It encodes underlying information that determines state transitions and rewards in the environment. For example, when one plays chess with a familiar opponent, the knowledge about the opponent's habits and preferences can be viewed as a belief state that may help to win the game rather than just looking at the chessboard. To solve PO tasks using a belief state, the belief state must contain information about the critical but unobservable state of the environment so that we can take the belief state together with raw observations as input of RL controllers. Unlike the approach using RNN as RL function approximators, the RL controllers here do not need to be recurrent since the belief state has already extracted the necessary information for making a decision at the current step. Perfectly-estimated belief states can thus be taken as "observations" of an RL agent that contains complete information for solving the task. Therefore, solving a PO task is segregated into a representation learning problem and a fully observable RL problem. Since fully observable RL problems have been well explored by the RL community, the critical challenge here is how to estimate the belief state, especially in a real-world application featured by stochastic state transition, context-dependence, and high-dimensional action and state space.

In this study, we developed a variational recurrent model (VRM) that models sequential observations and rewards using a latent stochastic variable. The VRM is an extension of the variational recurrent neural network (VRNN) model [34] that takes actions into account. Our approach takes the internal states of the VRM together with raw observations as the belief state. We then propose an algorithm to solve PO tasks by training the VRM and a feedforward RL controller network, respectively. The algorithm was designed for general POMDP problems by learning the representation of underlying states $s_t \in \mathbb{S}$ via predicting the subsequent observation and reward. It is expected to work in PO tasks in which s_t or $p(s_t)$ can be (at least partially) estimated from the history of observations $x_{1:t}$.

We experimentally evaluated the proposed algorithm in various PO robotic control tasks where no velocity information is observable, or only velocity information is observable, as well as the sequential target-reaching task introduced in the previous chapter (Chap .3.4.1). The agents showed substantial policy improvement in all tasks, and in some tasks, the algorithm performed essentially as in fully observable cases. In particular, our algorithm demonstrates greater performance compared to alternative approaches in the tasks that require long-term credit assignment, e.g., the

robotic control tasks where only velocity information is observable and the sequential target-reaching task.

Our contributions are shortly summarized as follows. First, we introduce a novel algorithm to handle POMDP problems by segregating representation learning and reinforcement learning. Second, we propose to take advantage of the recent variational RNN in SL for learning the belief state in POMDPs. Third, we propose a way to mitigate the representation shift problem by introducing the first-impression and keep-learning models (Chap 4.3.1). Fourth, we demonstrate how to perform sample-efficient RL in POMDP while significantly saving the computation cost of training the RNN models (Chap. 4.3.3). Finally, our algorithm is robust to the imperfectly-learned model by using both the belief states and the raw observations as actor and critic inputs (Chap. 4.5.8).

4.2 Related work

4.2.1 Deep RL for POMDP

Complexity-theoretic results show that controlling in POMDPs can be statistically and computationally intractable in general [152, 162, 235]. Therefore, many works focused on sub-domains of POMDPs. For example, meta-RL [182, 225, 239] can be considered a special case of POMDPs in which the agent needs to maximize average returns in a set of similar tasks that vary in reward function or state-transition dynamics. The varied task properties are invisible to the agent, making meta-RL a POMDP problem. However, meta-RL requires the agent to access a set of similar tasks, which usually cannot be satisfied in many real-world problems. Other relatively popular sub-domains are robust RL, where the learned policy should be robust to unobservable perturbations of the environmental properties [12, 113, 151], and generalization in RL, which pursues better performance in unseen testing environments [35, 204]. Rather than focusing on a particular sub-domain, our work aims at a "standard" POMDP, i.e., learning a reward-maximizing policy in a stationary POMDPs task [107].

To effectively perform RL in general POMDPs without prior knowledge, "*it is necessary to use memory of previous actions and observations to aid in the disambiguation of the states of the world*", as argued by Kaelbling et al. (Page 105, [107]). The most straightforward way is to simply concatenate all or a fixed length of the previous observations and actions as the input to RL functions [125, 144]. This way is not flexible if a long-term memory is needed for optimal decision and may face the difficulty of the curse of dimensionality [17].

The more common way of modeling the RL function with contextual information is to utilize RNNs to extract the features of previous observations and actions into the RNN states [1]. Most deep RL studies on solving memory-dependent POMDPS directly employed RNNs as the approximators for value and policy functions [103, 110, 181, 183, 234]. While this way is simple and often effective, it is unclear how representation

[1] As a side note, the Transformer model [231] can also be used for this purpose by modeling the state-transition trajectories as sequences [26, 104]. However, they are limited to offline RL as of now.

learning (extracting essential information from previous observations and actions) and RL (policy evaluation and policy improvement) affect each other.

In contrast, the proposed algorithm takes advantage of the belief states [82, 102, 107, 125] to separately handle representation learning and RL. While Ha & Schmidhuber's work [82] used a VAE to auto-encode the image observation into the belief state (context-free), our way of estimating belief states is context-dependent. In particular, we train a variational RNN world model to predict the next observation using previous observation and action. Then the RNN states are used as the belief states, which are then fed into the RL controllers. The work by Igl et al. [102] and Lee et al. [125] shared similar ideas with ours, which we will discuss in detail in Chap. 4.2.4.

4.2.2 Model-based RL

Typical model-based RL approaches utilize learned environment models for dreaming, i.e., generating imaginary state-transition data for training the agent [42, 82, 85, 108] or planning, i.e., inferring the policy that leads to a future trajectory with maximum returns [86, 111, 185]. The model-based RL methods usually require a well-designed and finely tuned model so that its predictions are accurate and robust. In our case, we do not use the VRMs for dreaming and planning but for encoding the belief states by learning the state transitions. We take advantage of a state-of-the-art model-free RL algorithm [83] to learn the policy using the belief states as additional information to the raw observations. Indeed, PO tasks can be solved without requiring VRMs to predict accurately (see Chap. 4.5.8). This distinguishes our algorithm from typical model-based RL methods.

4.2.3 Variational Bayes in RL

Variational Bayes (VB) methods have been widely used in deep learning [115] and RL. It is helpful to recognize the VB in RL studies in two aspects: using VB to derive a theoretical lower bound of a learning objective [55, 72, 128, 242] and using probabilistic network models based on VB (e.g., VAE [115]) as RL function approximators [82, 102, 125, 159, 252, 253].

Our study does not involve the former aspect, i.e., theoretical analysis with VB, but instead contributes by investigating how VB network models can benefit RL empirically. The usage of VB network models for DRL has gained attention from researchers recently. Ha & Schmidhuber [82] employed a VAE to encode the high-dimensional image observation into a lower-dimensional latent vector; Yin et al. [252] introduced a method to encourage exploration using intrinsic rewards based on the prediction error of a variational state-transition model. Okada et al. [159] utilized a deep VB state-transition model to perform efficient model-based planning in continuous control tasks. However, we use the variational RNN [34] for a different purpose—inferring the belief states in POMDPs.

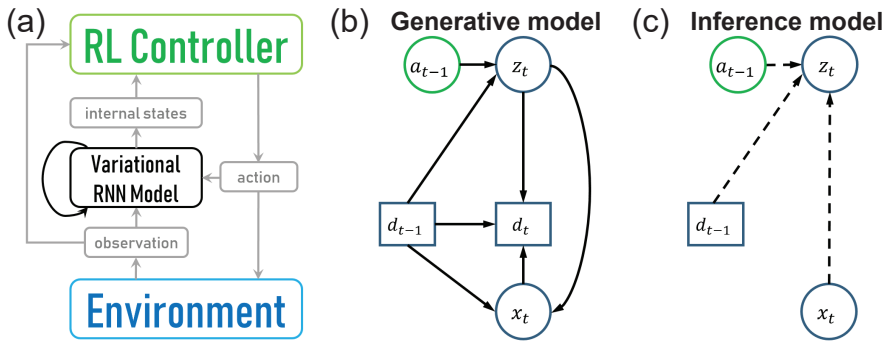


Figure 4.1: Diagrams of the proposed algorithm. **(a)** Overview. **(b, c)** The generative model and the inference model of a VRM.

4.2.4 Probabilistic models for encoding belief states in POMDPs

There also exist studies that share similar ideas with ours. The deep variational reinforcement learning (DVRL) algorithm [102] also modeled the state-transition function using variational RNNs to encode the belief states in POMDPs. However, there are key differences. First, DVRL trained the transition model and RL functions altogether by minimizing a combinatory RL and model prediction loss. In contrast, VRMs are trained separately from the RL controllers. Thus, our algorithm can save computation costs by reducing unnecessary updates of either of the two parts and avoid suffering from tuning the hyperparameters to trade-off between RL and model prediction losses. Also, DVRL employed a particle ensemble approach [198, 223] in training the model, whereas VRMs are trained simply by the reparameterization trick [115].

The work our algorithm most closely resembles is known as *stochastic latent actor-critic* (SLAC) [125][2], in which a latent variable model for observation prediction was trained. The latent variable was used as the belief state for the critic (but not for the actor). SLAC showed promising results using pixels-based robotic control tasks, where velocity information needs to be inferred from third-person images of the robot. However, there are important distinctions between our work and SLAC. Here we consider more general PO environments in which the reward may depend on a long history of inputs. The actor-network of SLAC did not take advantage of the latent variable. Instead, it used some steps of raw observations as input, which creates problems in achieving long-term memorization of reward-related state transitions. Furthermore, unlike VRM, SLAC did not include raw observations in the input of the critic, which may complicate training the critic before the model converges.

4.3 Methods

4.3.1 Variational recurrent state-transition models

An overall diagram of the proposed algorithm is summarized in Fig. 4.1(a), while a more detailed computational graph is plotted in Fig. 4.2. The core idea is to ease the learning

[2]This work and SLAC were concurrent.

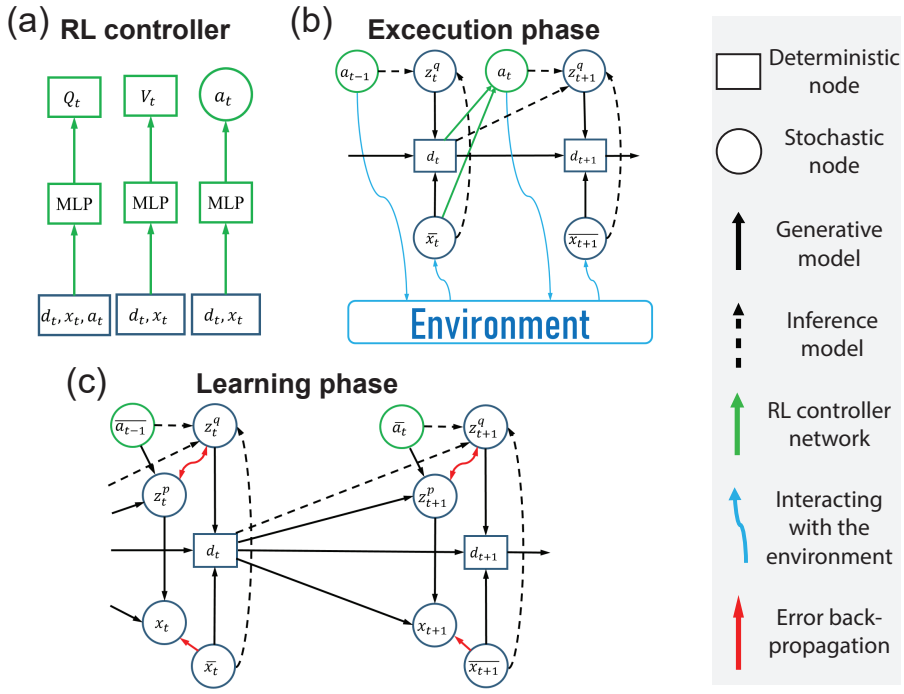


Figure 4.2: Computation diagram of the proposed algorithm. **(a)** The RL controller. **(b)** The execution phase. **(c)** The learning phase of a VRM. $\boldsymbol{a}$: action; $\boldsymbol{z}$: latent variable; $\boldsymbol{d}$: RNN state variable; $\boldsymbol{x}$: raw observation (including reward); Q: state-action value function; V: state value function. A bar on a variable means that it is the actual value from the replay buffer or the environment. Each stochastic variable follows a parameterized diagonal Gaussian distribution.

by separately handling representation learning and RL. The representation learning part is achieved by training the VRM to predict the next observation using a sequence of observations and actions. By doing so, the internal states of the VRM should encode the underlying environmental states (as much as possible) that are important to state transitions. Thus, providing the VRM's internal states to the RL controllers reduces the burden on RL.

For representation learning, we extend the original VRNN model [34] to the proposed VRM model by adding action feedback, i.e., actions taken by the agent are used in the inference model and the generative model. Also, since we are modeling state-transition and reward functions, we include the reward r_{t-1} in the current raw observation $\boldsymbol{x}_t$ for convenience. Thus, we have the inference model (Fig. 4.1(c)), denoted by ϕ, as

$$\boldsymbol{z}_{\phi,t}|\boldsymbol{x}_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{\phi,t}, \mathrm{diag}(\boldsymbol{\sigma}^2_{\phi,t})\right), \text{ where } \left[\boldsymbol{\mu}_{\phi,t}, \boldsymbol{\sigma}^2_{\phi,t}\right] = \phi(\boldsymbol{x}_t, \boldsymbol{d}_{t-1}, \boldsymbol{a}_{t-1}), \tag{4.1}$$

The generative model (Fig. 4.1(b)), denoted by θ here, is

$$\begin{aligned} \boldsymbol{z}_t &\sim \mathcal{N}\left(\boldsymbol{\mu}_{\theta,t}, \mathrm{diag}(\boldsymbol{\sigma}^2_{\theta,t})\right), \quad \left[\boldsymbol{\mu}_{\theta,t}, \boldsymbol{\sigma}^2_{\theta,t}\right] = \theta^{\mathrm{prior}}(\boldsymbol{d}_{t-1}, \boldsymbol{a}_{t-1}), \\ \boldsymbol{x}_t|\boldsymbol{z}_t &\sim \mathcal{N}\left(\boldsymbol{\mu}_{x,t}, \mathrm{diag}(\boldsymbol{\sigma}^2_{x,t})\right), \quad \left[\boldsymbol{\mu}_{x,t}, \boldsymbol{\sigma}^2_{x,t}\right] = \theta^{\mathrm{decoder}}(\boldsymbol{z}_t, \boldsymbol{d}_{t-1}). \end{aligned} \tag{4.2}$$

For building recurrent connections, the choice of RNN types is not limited. In our study, the *long-short term memory* (LSTM) [98] is used since it works well in general cases. So we have $\boldsymbol{d}_t = \text{LSTM}(\boldsymbol{d}_{t-1}; \boldsymbol{z}_t, \boldsymbol{x}_t)$.

As in training a VRNN, the VRM is trained by maximizing the evidence lower bound (Fig. 4.1(c))

$$\begin{aligned} ELBO = \sum_t \big\{ & \mathbb{E}_{q_\phi} \left[\log p_\theta(\boldsymbol{x}_t | \boldsymbol{z}_{1:t}, \boldsymbol{x}_{1:t-1}) \right] \\ & - D_{KL} \left[q_\phi(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, \bar{\boldsymbol{x}}_{1:t}, \bar{\boldsymbol{a}}_{1:t}) || p_\theta(\boldsymbol{z}_t | \boldsymbol{z}_{1:t-1}, \bar{\boldsymbol{x}}_{1:t-1}, \bar{\boldsymbol{a}}_{1:t}) \right] \big\} . \end{aligned} \tag{4.3}$$

In practice, the first term $\mathbb{E}_{q_\phi} \left[\log p_\theta(\boldsymbol{x}_t | \boldsymbol{z}_{1:t}, \boldsymbol{x}_{1:t-1}) \right]$ can be obtained by unrolling the RNN using the inference model (Fig. 4.1(c)) with sampled sequences of $\boldsymbol{x}_t$. Since q_ϕ and p_θ are parameterized Gaussian distributions, the KL-divergence term can be analytically expressed as

$$D_{KL} \left[q_\phi(\boldsymbol{z}_t) || p_\theta(\boldsymbol{z}_t) \right] = \log \frac{\boldsymbol{\sigma}_{\phi,t}}{\boldsymbol{\sigma}_{\theta,t}} + \frac{(\boldsymbol{\mu}_{\phi,t} - \boldsymbol{\mu}_{\theta,t})^2 + \boldsymbol{\sigma}_{\phi,t}^2}{2\boldsymbol{\sigma}_{\theta,t}^2} - \frac{1}{2}. \tag{4.4}$$

For computation efficiency in experience replay, we train a VRM by sampling mini-batches of truncated sequences of fixed length instead of whole episodes.

Since the training of a VRM is segregated from the training of the RL controllers, there are several strategies for conducting them in parallel. For the RL controller, we adopted a smooth update strategy as in SAC [83], i.e., performing experience replay once every n steps. To train the VRM, one can also conduct smooth updates. However, in that case, RL suffers from instability of the representation of underlying states in the VRM before it converges. Also, the stochasticity of RNN state variables $\boldsymbol{d}$ can be meaninglessly high at the early stage of training, which may create problems in RL. Another strategy is to pre-train the VRM for abundant epochs only before RL starts, which, unfortunately, can fail if novel observations from the environment appear after some degree of policy improvement. Moreover, if pre-training and smooth update are both applied to the VRM, RL may suffer from a large representation shift of the belief state.

To resolve this conflict, we propose using two VRMs, which we call the *first-impression model* and the *keep-learning model*, respectively. As the names suggest, we pre-train the first-impression model and stop updating it when RL controllers and the keep-learning model start smooth updates. Then we take state variables from both VRMs, together with raw observations, as the input for the RL controller. We found that this method yields better overall performance than using a single VRM (Chap. 4.5.6).

4.3.2 Reinforcement learning controllers

Since the VRM takes the responsibility of extracting the underlying environmental states from a sequence of observations and actions, the RL controllers can focus on RL by including the belief states (the VRM's internal states), in addition to the current raw observation, as the input to value and policy functions. Then, using feedforward neural networks for the RL controllers can avoid the difficulties of training RNNs [166].

In particular, we use multi-layer perceptrons (MLP) as function approximators for V, Q, respectively(Fig. 4.1(a)). Inputs for the Q_t network are $(\boldsymbol{x}_t, \boldsymbol{d}_t, \boldsymbol{a}_t)$, and V_t is mapped from $(\boldsymbol{x}_t, \boldsymbol{d}_t)$. Following SAC [83], we use two Q-networks λ_1 and λ_2 and compute $Q = \min(Q_{\lambda_1}, Q_{\lambda_2})$ in Eq. 2.20 and 2.22 for better performance and stability. Furthermore, we also used a target value network for computing V in Eq. 2.21 as in SAC [83]. The policy function π_η follows a parameterized Gaussian distribution $\mathcal{N}\left(\boldsymbol{\mu}_\eta(\boldsymbol{d}_t, \boldsymbol{x}_t), diag\left(\boldsymbol{\sigma}_\eta(\boldsymbol{d}_t, \boldsymbol{x}_t)\right)\right)$ where $\boldsymbol{\mu}_\eta$ and $\boldsymbol{\sigma}_\eta$ are also MLPs.

In the execution phase (Fig. 4.1(b)), observation and reward $\boldsymbol{x}_t = (\boldsymbol{X}_t, r_{t-1})$ are received as VRM inputs to compute internal states $\boldsymbol{d}_t$ using inference models. Then, the agent selects an action, sampled from $\pi_\eta(\boldsymbol{a}_t|\boldsymbol{d}_t, \boldsymbol{x}_t)$, to interact with the environment.

To train RL networks, we first sample sequences of steps from the replay buffer as minibatches; thus, $\boldsymbol{d}_t$ can be computed by the inference models using recorded observations $\bar{\boldsymbol{x}}_t$ and actions $\bar{\boldsymbol{a}}_t$ (See Chap. 4.3.4). Then RL networks are updated by minimizing the loss functions with gradient descent. Gradients stop at $\boldsymbol{d}_t$ so that training of RL networks does not involve updating VRMs.

4.3.3 Update-to-data ratio

There is another notable benefit of training the keep-learning VRM and RL controllers separately: we can use different update-to-data ratios [28] for the two parts. In particular, we perform 1 gradient step for the keep-learning VRM and 5 for RL controllers per 5 environment steps (5 samples). This is reasonable because the learning rates for RL are usually small (e.g., 3e-4 in SAC and TD3 [70, 83]) so as to maintain stable training, while in supervised learning without such difficulties, the learning rate can be larger (e.g., 1e-3 in the original paper of VRNN [34]). As such, the computational cost of training the VRMs is largely reduced, and thus, the whole training time is significantly saved due to the fact that the training of RNNs (VRMs) is much more costly than that of MLPs (RL controllers).

4.3.4 Implementation details

In this subsection, we describe the details of implementing our algorithm as well as the alternative ones.

Network architectures

The first-impression model and the keep-learning model adopted the same architecture. Size of $\boldsymbol{d}$ and $\boldsymbol{z}$ is 256 and 64, respectively. We used one-hidden-layer fully-connected networks with 128 hidden neurons for the inference models $\left[\boldsymbol{\mu}_{\phi,t}, \boldsymbol{\sigma}^2_{\phi,t}\right] = \phi(\boldsymbol{x}_t, \boldsymbol{d}_{t-1}, \boldsymbol{a}_{t-1})$, as well as for $\left[\boldsymbol{\mu}_{\theta,t}, \boldsymbol{\sigma}^2_{\theta,t}\right] = \theta^{\text{prior}}(\boldsymbol{d}_{t-1}, \boldsymbol{a}_{t-1})$ in the generative models. For the decoder $\left[\boldsymbol{\mu}_{x,t}, \boldsymbol{\sigma}^2_{x,t}\right] = \theta^{\text{decoder}}(\boldsymbol{z}_t, \boldsymbol{d}_{t-1})$ in the generative models, we used 2-layers MLPs with 128 neurons in each layer. The input processing layer f_x is also an one-layer MLP with size-128. For all the Gaussian variables, output functions for mean are linear and output functions for variance are softplus. Other activation functions of the VRMs are tanh.

We used soft actor-critic (SAC, Chap. 2.1.10) as the baseline RL algorithm. SAC was originally developed for fully observable environments; thus, the raw observation at the current step $\boldsymbol{x}_t$ was used as network input. In this work, we apply SAC in PO tasks by including the state variable $\boldsymbol{d}_t$ of the VRNN in the input of function approximators of both the actor and the critic.

Initial states of the VRMs

To train the VRMs, one can use a number of entire episodes as a minibatch, using zero initial states, as in Heess et al.'s work [94]. However, when tackling long episodes (e.g., there can be 1,000 steps in each episode in the robotic control tasks we used) or even infinite-horizon problems, the computation consumption will be huge in back-propagation through time (BPTT). For better computation efficiency, we used 4 length-64 sequences for training the RNNs, and applied the *burn-in* method for providing the initial states [110], or more specifically, unrolling the RNNs using a portion of the replay sequence (burn-in period, up to 64 steps in our case) from zero initial states. We assume that proper initial states can be obtained in this way. This is crucial for the tasks that require long-term memorization and is helpful in reducing bias introduced by incorrect initial states in general cases.

4.3.5 Hyperparameters

For the RL controllers, we adopted hyperparameters from the original SAC implementation [84]. Both the keep-learning and first-impression VRMs were trained using a learning rate of 0.0008. We pre-trained the first-impression VRM for 5,000 epochs and updated the keep-learning VRM every 5 steps. Batches of size 4, each containing a sequence of 64 steps, were used for training both the VRMs and the RL controllers. All tasks used the same hyperparameters (Chap. 4.3.4). Summaries of hyperparameters can be found in Table 4.1 and 4.2.

Table 4.1: Shared hyperparameters for all the algorithms and tasks in the study, adopted from the original SAC implementation [84]. The degree of freedom (DOF) of each environment is listed in Table 4.3.

Hyperparameter	Description	Value
γ	Discount factor	0.99
step_start_RL	From how many steps to start training the RL controllers	1,000
train_interval_RL	Interval of training the RL controllers	1
lr_actor	Learning rate for the actor	0.0003
lr_critic	Learning rate for the critic	0.0003
lr_α	Learning rate for the entropy coefficient α	0.0003
$\mathcal{H}_{\text{tar}}$	Target entropy	$-$DOF
optimizer	Optimizers for all the networks	Adam [114]
τ	Fraction of updating the target network each gradient step	0.005
policy_layers	MLP layer sizes for $\boldsymbol{\mu}_\eta$ and $\boldsymbol{\pi}_\eta$	256, 256
value_layers	MLP layer sizes for V_ϕ and Q_λ	256, 256

Table 4.2: Hyperparameters for the proposed algorithm.

Hyperparameter	Description	Value
train_times_FIVRM	Epoches of training the first-impression model.	5,000
train_interval_KLVRM	Interval of training the keep-learning model.	5
lr_model	Learning rate for the VRMs	0.0008
seq_len	How many steps in a sampled sequence for each update	64
batch_size	How many sequences to sample for each update	4

4.4 Environments

We used environments (and modified them for PO versions) from OpenAI Gym [24]. A list of used environments can be found in Table 4.3. The CartPole environment with a continuous action space was from Danforth's GitHub repository [38], and the codes for the sequential target reaching tasks were provided by the authors [87].

Table 4.3: Information of the environments we used.

Name	dim($\mathbb{X}$)	DOF	Maximum steps
Pendulum	3	1	200
Pendulum (velocities only)	1	1	200
Pendulum (no velocities)	2	1	200
CartPole	4	1	1,000
CartPole (velocities only)	2	1	1,000
CartPole (no velocities)	2	1	1,000
RoboschoolHopper	15	3	1,000
RoboschoolHopper (velocities only)	6	3	1,000
RoboschoolHopper (no velocities)	9	3	1,000
RoboschoolWalker2d	22	6	1,000
RoboschoolWalker2d (velocities only)	9	6	1,000
RoboschoolWalker2d (no velocities)	13	6	1,000
RoboschoolAnt	28	8	1,000
RoboschoolAnt (velocities only)	11	8	1,000
RoboschoolAnt (no velocities)	17	8	1,000
Sequential goal reaching task	12	2	128

4.5 Results

To empirically evaluate our algorithm, we performed experiments in a range of (partially observable) continuous control tasks and compared it to the following alternative algorithms. The overall procedure is summarized in Algorithm 4.

Algorithm 4 Variational Recurrent Models with Soft Actor Critic

Initialize the first-impression VRM $\mathcal{M}_f$ and the keep-learning VRM $\mathcal{M}_k$, the RL controller $\mathcal{C}$, and the replay buffer $\mathcal{D}$, global step $t \leftarrow 0$.
repeat
  Initialize an episode, assign $\mathcal{M}$ with zero initial states.
  while episode not terminated **do**
    Sample an action $\boldsymbol{a}_t$ from $\pi(\boldsymbol{a}_t|\boldsymbol{d}_t, \boldsymbol{x}_t)$ and execute a_t, $t \leftarrow t+1$.
    Record $(\boldsymbol{x}_t, \boldsymbol{a}_t, done_t)$ into $\mathcal{B}$.
    Compute 1-step forward of both VRMs using inference models.
    if $t == step\_start\_RL$ **then**
      For N epochs, sample a minibatch from $\mathcal{B}$ to update $\mathcal{M}_f$ (Eq. 4.3).
    end if
    if $t > step\_start\_RL$ and $mod(t, train\_interval\_KLVRM) == 0$ **then**
      Sample a minibatch from $\mathcal{B}$ to update $\mathcal{M}_k$ (Eq. 2.20, 2.21, 2.22, 2.23) .
    end if
    if $t > step\_start\_RL$ and $mod(t, train\_interval\_RL) == 0$ **then**
      Sample a minibatch from $\mathcal{B}$ to update $\mathcal{R}$ (Eq. 4.3) .
    end if
  end while
until training stopped

4.5.1 Alternative algorithms

- **SAC-MLP**: The vanilla soft actor-critic implementation [83, 84] (Chap. 2.1.10), in which each function is approximated by a 2-layer MLP taking raw observations as input. We followed the original implementation of SAC [83] including hyperparameters. We also applied automatic learning of the entropy coefficient α (inverse of the reward scale [83]) as introduced by the Haarnoja et al.'s newer version [84] to avoid tuning the reward scale for each task.

- **SAC-LSTM**: Soft actor-critic with recurrent networks as function approximators, where raw observations are processed through a standard LSTM layer (no latent stochasticity) followed by 2 layers of MLPs. This allows the agent to make decisions based on the whole history of raw observations. In this case, the network has to conduct representation learning and dynamic programming collectively. Our algorithm is compared with SAC-LSTM to demonstrate the effect of separating representation learning from dynamic programming. To apply recurrency to SAC's function approximators, we added an LSTM network with size-256 receiving raw observations as input. The function approximators of actor and critic were the same as those in SAC except for receiving the LSTM's output as input. The gradients could pass through the LSTM so that the LSTM and MLPs were trained as a whole.

- **SLAC**: The stochastic latent actor-critic algorithm introduced by Lee et al. [125], which is a state-of-the-art RL algorithm for solving POMDP tasks. It was shown that SLAC outperformed other model-based and model-free algorithms [86, 102],

in robotic control tasks with a third-person image of the robot as observation[3]. We mostly followed the implementation of SLAC explained in the original paper [125]. One modification is that since their work used pixels as observations, convolutional neural networks (CNN) and transposed CNNs were chosen for input feature extracting and output decoding layers; in our case, we replaced the CNN and transposed CNNs with 2-layers MLPs with 256 units in each layer. In addition, the authors set the output variance $\sigma_{y,t}^2$ for each image pixel as 0.1. However, $\sigma_{y,t}^2 = 0.1$ can be too large for joint states/velocities as observations. We found that it will lead to better performance by setting $\boldsymbol{\sigma}_{y,t}$ as trainable parameters (as that in our algorithm). We also used a 2-layer MLP with 256 units for approximating $\boldsymbol{\sigma}_y(\boldsymbol{x}_t, \boldsymbol{d}_{t-1})$. To avoid network weights being divergent, all the activation functions of the model were tanh except those for outputs.

Note that in our algorithm, we apply pre-training of the first-impression model. For a fair comparison, we also perform pre-training for the alternative algorithm with the same epochs. For SAC-MLP and SAC-LSTM, pre-training is conducted on RL networks; while for SLAC, its model is pre-trained.

4.5.2 Partially observable classic control tasks

The *Pendulum* and *CartPole* [13] tasks are the classic control tasks for evaluating RL algorithms (Fig. 4.3, Left). The CartPole task requires learning a policy that prevents the pole from falling down and keeps the cart from running away by applying a (1-dimensional) force to the cart, in which observable information is the coordinate of the cart, the angle of the pole, and their derivatives w.r.t time (i.e., velocities). For the Pendulum task, the agent needs to learn a policy to swing an inverse pendulum up and maintain it at the highest position in order to obtain more rewards.

We are interested in classic control tasks because they are relatively easy to solve when fully observable, and thus the PO cases can highlight the representation learning problem. Experiments were performed in these two tasks, as well as their PO versions, in which either velocities cannot be observed, or only velocities can be observed. The latter case is meaningful in real-life applications because an agent may not be able to perceive its own position, but it can estimate its speed.

As expected, SAC-MLP failed to solve the PO tasks (Fig. 4.3). While our algorithm succeeded in learning to solve all these tasks, SAC-LSTM showed poorer performance in some of them. In particular, in the pendulum task with only angular velocity observable, SAC-LSTM may suffer from the periodicity of the angle. SLAC performed well in the CartPole tasks but showed less satisfactory sample efficiency in the Pendulum tasks.

[3]SLAC was developed for pixel observations. To compare it with our algorithm, we made some modifications to its implementation. Nonetheless, we expect the comparison can demonstrate the effect of the key differences (Chap. 4.2.4)

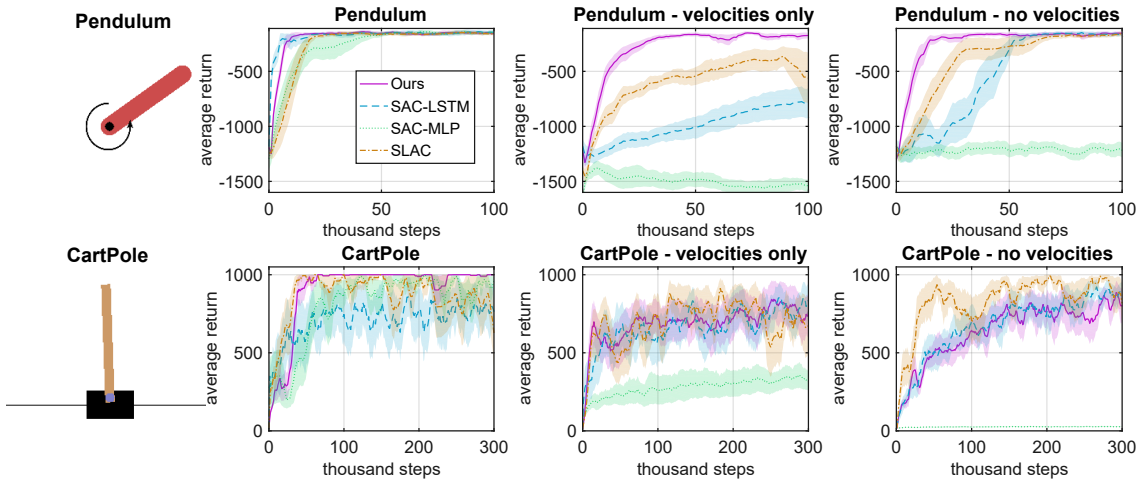


Figure 4.3: Learning curves of the classic control tasks. Shaded areas indicate SEM. The average return is defined as the total rewards in an episode on average for each agent. For the Pendulum task, the reward is always no larger than zero and is closer to zero if the pendulum is closer to the top. The reward for CartPole is 1 at each step, i.e., the total rewards are the steps of keeping balance (maximum 1,000 in an episode).

4.5.3 Partially observable robotic control tasks

To examine the performance of the proposed algorithm in more challenging control tasks with higher degrees of freedom (DOF), we also evaluated the performance of the proposed algorithm in the OpenAI *Roboschool* environments [24]. The Roboschool environments include a number of continuous robotic control tasks, such as teaching a multiple-joint robot to walk as fast as possible without falling down (Fig. 4.4, Left). The original Roboschool environments are nearly fully observable since observations include the robot's coordinates and (trigonometric functions of) joint angles, as well as (angular and coordinate) velocities. As in the PO classic control tasks, we also performed experiments in the PO versions of the Roboschool environments.

Using our algorithm, experimental results (Fig. 4.4) demonstrated substantial policy improvement in all PO tasks. In some PO cases, the agents achieved comparable performance to that in fully observable cases. For tasks with unobserved velocities, our algorithm performed similarly to SAC-LSTM. This is because velocities can be simply estimated by one-step differences in robot coordinates and joint angles, which eases representation learning. However, in environments where only velocities can be observed, our algorithm significantly outperformed SAC-LSTM, presumably because SAC-LSTM is less efficient at encoding underlying states from velocity observations. Also, we found that learning of a SLAC agent was unstable, i.e., it sometimes could acquire a near-optimal policy, but often its policy converged to a poor one. Thus, the average performance of SLAC was less promising than ours in most of the PO robotic control tasks.

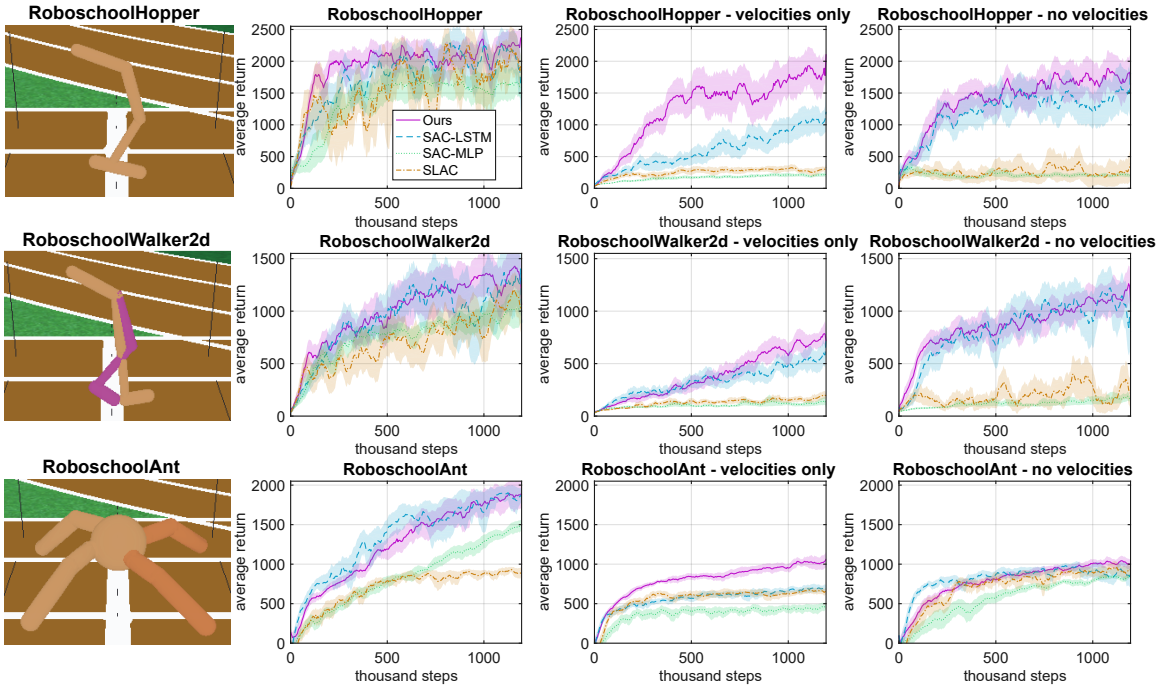


Figure 4.4: Learning curves of the robotic control tasks, plotted in the same way as in Fig. 4.3. The average return is defined as the total rewards in an episode on average for each agent, where the reward functions are unchanged from the Roboschool environments.

4.5.4 Long-term memorization tasks

Another common type of PO task requires long-term memorization of past events. To solve these tasks, an agent needs to learn to extract and remember critical information from the whole history of raw observations. Therefore, we also examined our algorithm and other alternatives in a long-term memorization task, the sequential target reaching task [87], in which a robot agent needs to reach 3 different targets in a certain sequence (Fig. 4.5, Left, see Chap. 3.4.1 for more details). The robot can control its two wheels to move or turn and will get one-step small, medium, and large rewards when it reaches the first, second, and third targets, respectively, in the correct sequence. The robot senses distances and angles from the 3 targets, but does not receive any signal indicating which target to reach. In each episode, the robot's initial position and those of the three targets are randomly initialized. In order to obtain rewards, the agent needs to infer the correct target using historical observations.

We found that agents using our algorithm achieved almost 100% success rate (defined in the same way as in Chap. 3.4.5, i.e., reaching 3 targets in the correct sequence within 50 steps). SAC-LSTM also achieved a similar success rate after convergence but spent more training steps learning to encode underlying goal-related information from sequential observations. Also, SLAC struggled hard to solve this task since its actor only received limited steps of observations, making it difficult to infer the correct target.

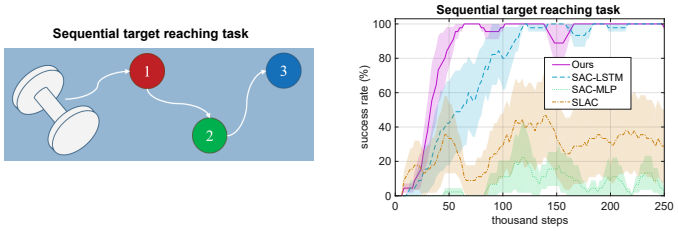


Figure 4.5: Learning curves of the sequential target reaching task.

4.5.5 Convergence of the keep-learning VRM

One of the most concerning problems of our algorithm is that input of the RL controllers can experience representation change because the keep-learning model is not guaranteed to converge if novel observation appears due to improved policy (e.g., for a hopper robot, "in-the-air" state can only happen after it learns to hop). To empirically investigate how the convergence of the keep-learning VRM affects policy improvement, we plot the loss functions (negative ELBOs) of the keep-learning VRM for 3 example tasks (Fig. 4.6). For a simpler task (CartPole), the policy was already near-optimal before the VRM fully converged. We also saw that the policy was gradually improved after the VRM mostly converged (RoboschoolAnt - no velocities). And the policy and the VRM were being improved in parallel (RoboschoolAnt - velocities only).

The results suggested that policy could be improved with sufficient sample efficiency even if the keep-learning VRM did not converge. This can be explained that the RL controller also extracts information from the first-impression model and the raw observations, which did not experience representation change during RL. Indeed, our ablation study showed performance degradation in many tasks without the first-impression VRM (Chap. 4.5.6).

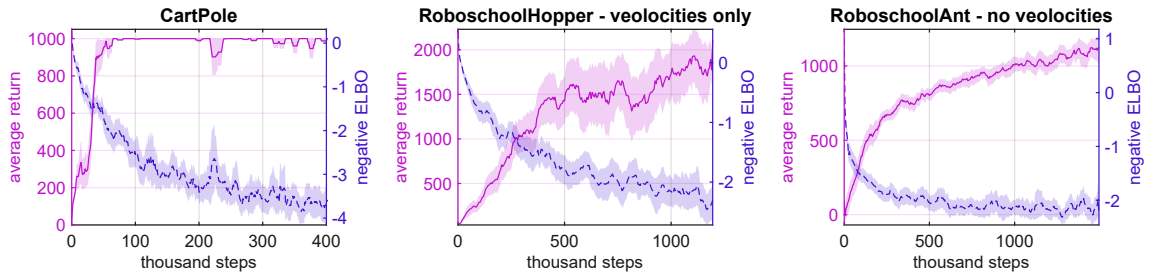


Figure 4.6: Example tasks showing the relationship between average return of the agent and negative ELBO (loss function, dashed) of the keep-learning VRM.

4.5.6 Ablation study

This section demonstrated an ablation studies in which we compared the performance of the proposed algorithm to the same but with some modifications:

- **With a single VRM**. In this case, we used only one VRM and applied both pre-training and smooth updates to it.

- **Only first-impression model**. In this case, only the first-impression model was used and pre-trained.

- **Only keep-learning model**. In this case, only the keep-learning model was used, and smooth-update was applied.

- **Deterministic model**. In this case, the first-impression model and the keep-learning model were deterministic RNNs that learned to model the state transitions by minimizing the mean-square error between prediction and observations instead of $ELBO$. The network architecture was mostly the same as the VRM expect that the inference model and the generative model were merged into a deterministic one.

The learning curves are shown in Fig. 4.7. It can be seen that the proposed algorithm consistently performed similarly or better than the modified ones.

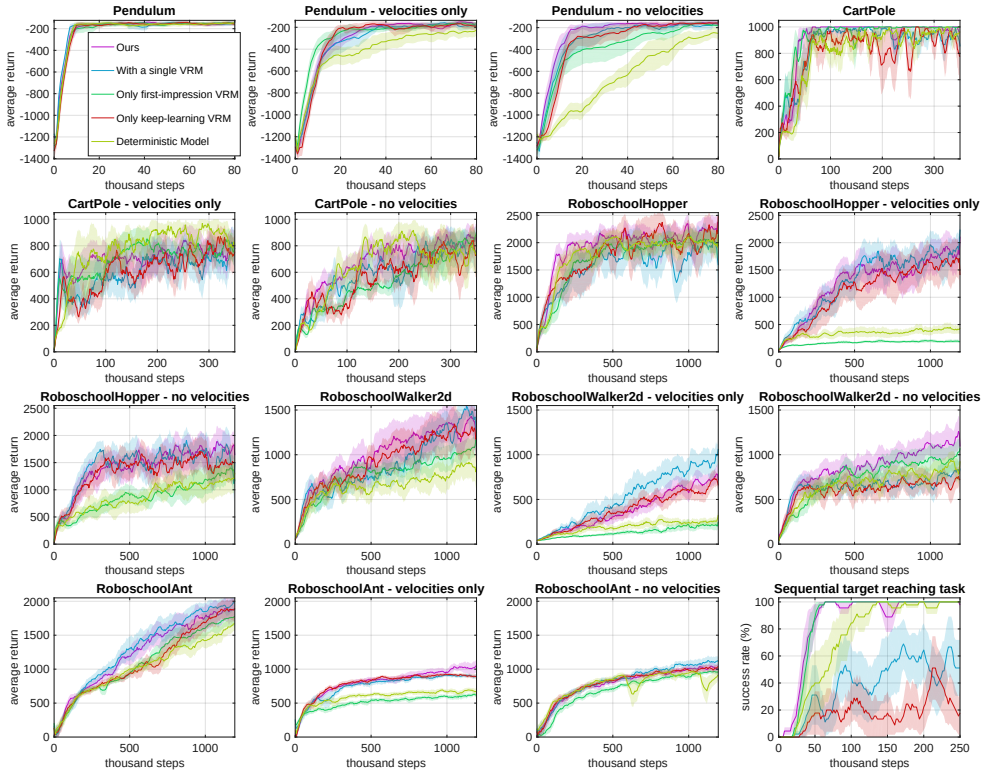


Figure 4.7: Learning curves of our algorithms and the modified ones.

4.5.7 Visualization of trained agents

Here we show the actual movements of the trained robots in the PO robotic control tasks (Fig. 4.8). It can be seen that the robots succeeded in learning to hop or walk, although their policy may be sub-optimal.

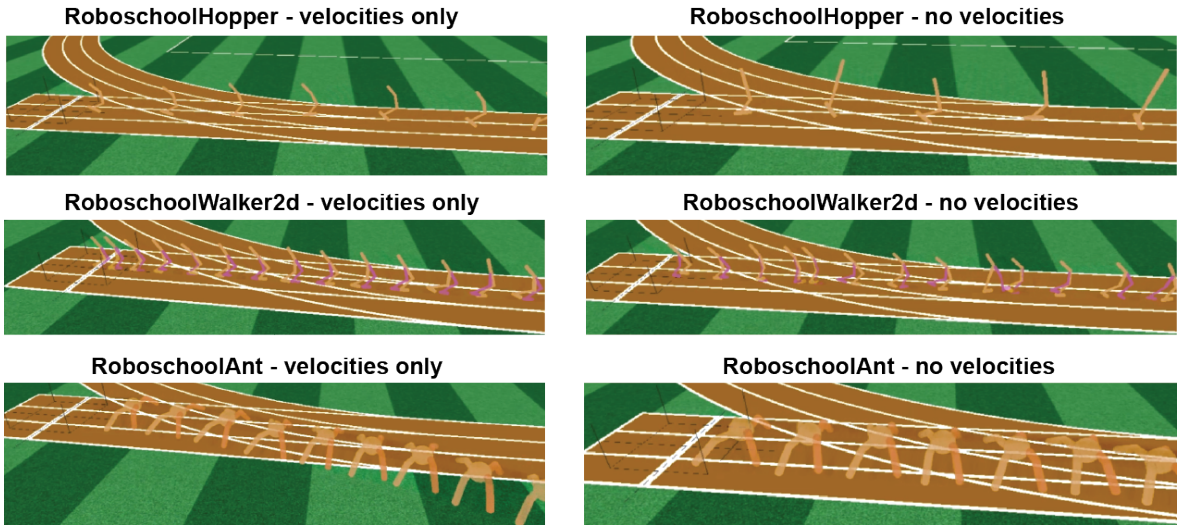


Figure 4.8: Robots learned to hop or walk in PO environments using our algorithm. Each panel shows the trajectory of a trained agent (randomly selected) within one episode.

4.5.8 Model accuracy

Our algorithm relies mostly on the encoding capacity of models but does not require models to make accurate predictions of future observations. Fig. 4.9 shows open-loop (using the inference model to compute the latent variable z) and close-loop (purely using the generative model) prediction of raw observation by the keep-learning models of randomly selected trained agents. Here we showcase "RoboschoolHopper - velocities only" and "Pendulum - no velocities" because in these tasks, our algorithm achieved similar performance to those in fully-observable versions (Fig. 4.4), despite that the prediction accuracy of the models was imperfect.

4.5.9 Sensitivity to hyperparameters of the VRMs

To empirically show how the choice of hyperparameters of the VRMs affects RL performance, we conducted experiments using hyperparameters different from those used in the main study. More specifically, the learning rate for both VRMs was randomly selected from {0.0004, 0.0006, 0.0008, 0.001} and the sequence length was randomly selected from {16, 32, 64} (the batch size was $256/(sequence\_length)$ to ensure that the total number of samples in a batch was 256 which matched with the alternative approaches). The other hyperparameters were unchanged.

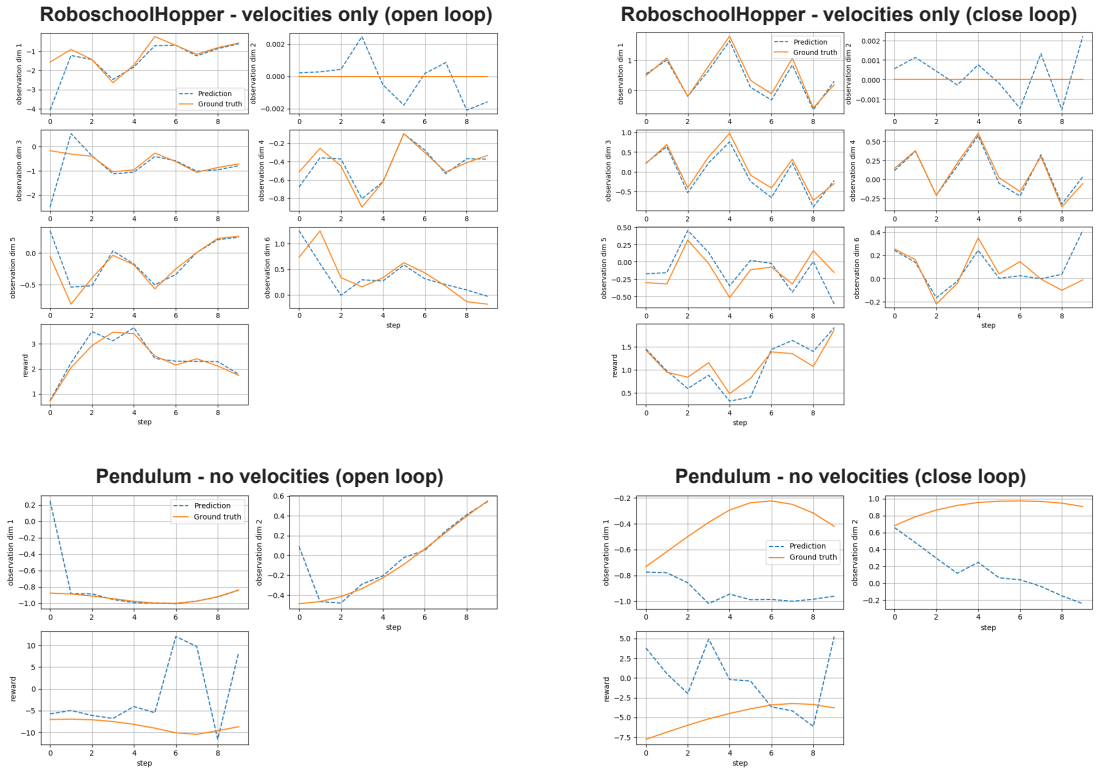


Figure 4.9: Examples of observation predictions by keep-learning VRMs of trained agents.

The results can be checked in Fig 4.10 for all the environments we used. The overall performance did not significantly change using different, random hyperparameters of the VRMs, although we could observe significant performance improvement (e.g., RoboshoolWalker2d) or degradation (e.g., RoboshoolHopper - velocities only) in a few tasks using different hyperparameters. Therefore, the representation learning part (VRMs) of our algorithm does not suffer from high sensitivity to hyperparameters. This can be explained by the fact that we do not use a bootstrapping (e.g., the estimation of targets of value functions depends on the estimation of value functions) update rule [208] to train the VRMs.

4.6 Summary

This study proposes a variational recurrent model for learning to represent underlying states of PO environments and the corresponding algorithm for solving POMDPs. Our experimental results demonstrate the effectiveness of the proposed algorithm in tasks in which underlying states cannot be simply inferred using a sequence of observations.

The first contribution of our work is that we point out the importance of carefully handling the representation learning problem in POMDPs. Our work is one of the earliest attempts[4] to empirically investigate the performance of deep RL agents that

[4] A contemporary study is from Lee et al. [125], and a later one is from Ota et al. [161]

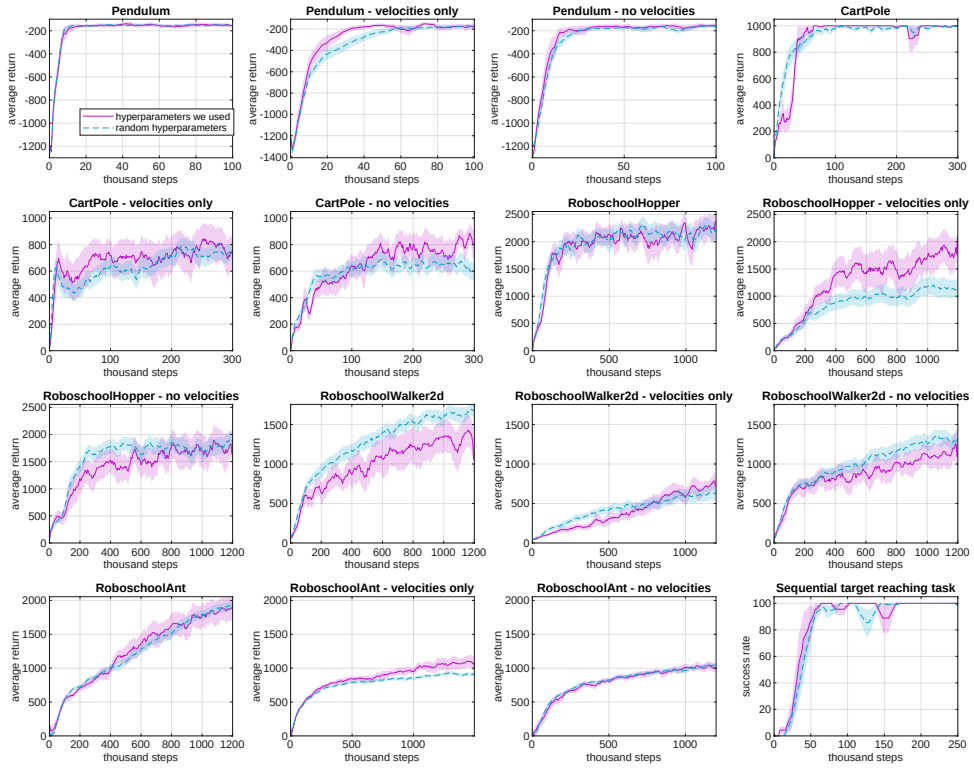


Figure 4.10: The learning curves of our algorithm using the hyperparameters for the VRMs used in the study (Table 4.2) and using a range of random hyperparameters. Data are Mean ± SEM, obtained from 20 repeats using different random seeds.

detach representation learning from RL for solving POMDP tasks. The results on various PO tasks show that the proposed method outperforms the traditional method that straightforwardly uses an RNN for RL function approximation. Future works are expected to develop more advanced methodologies for handling representation learning in RL.

We propose to employ the VRNN structure [34] which is advantageous in SL, for representation learning. The VRNN learns to predict the subsequent observation, given the history of observations and actions. By doing so, the internal state of the VRNN learns to encode the underlying environmental states that are critical to state transitions. An important result is that the usage of the VRNN structure enhances RL performance compared to using deterministic RNNs (Chap. 4.5.6). A potential reason to account for this result is that the network stochasticity enhances the smoothness and robustness of the learned representation [34, 115]. This result indicates that stochastic models [5, 34, 78, 115] can be beneficial to representation learning problems in RL even when the environmental dynamics are deterministic.

Moreover, we propose a heuristic approach to mitigate the representation shift problem, which occurs when new state transitions come with updated policy (Chap 4.3.1). We employ two VRMs with different learning schemes: the first-impression VRM,

which is trained using the initial exploration experiences and is then fixed, and the keep-learning VRM, which is continuously being trained during the entire learning course. While we show that this approach enhances RL performance in the proposed tasks (Chap. 4.5.6, this idea should also apply to more general cases of representation learning in online RL where the representation shift problem occurs.

In addition, we introduced a way to save the computation cost of training the RNN models (Chap. 4.3.3). Since the representation learning part can be considered as (self-)supervised learning (thus avoiding the difficulties in RL such as bootstrapping), we can apply a larger learning rate and update the VRM fewer times. This technical trick greatly reduces the time to train the RNNs, which is the major bottleneck of computation time.

Finally, unlike typical model-based RL methods that demand accurate world models to perform dreaming or planning, our algorithm is relatively robust to imperfectly-learned models (Chap. 4.5.8). The primary reason is that our algorithm uses both the learned belief state and the raw observation as the inputs to the RL functions. Thus, even if the VRM has not perfectly learned the belief state, the policy function may output reasonably good actions based on the raw observation and the essential contextual information encoded in the belief state (e.g., which target to reach in the sequential target-reaching task).

4.7 Discussion

4.7.1 Model-based and model-free RL

Traditionally, it was considered that there are two distinct systems for model-based and model-free RL in the brain [76, 126]. Several neuroscientific studies investigated the brain mechanisms underlying the arbitration between model-based and model-free RL [126, 200]. They usually consider a simple ensemble of the model-based and model-free policies for the outcome action, e.g., the outcome action is modeled via a linear combination of two Q-functions obtained by model-based and model-free RL, respectively [126]. However, the assumption that model-based and model-free RL in the brain are two totally distinct systems might be overly-simplified. One brain region may contribute to both model-based and model-free RL. For example, Stachenfeld et al. [205] suggested that the hippocampus can learn a *successor representation* of the environment that benefits both model-free and model-based RL.

Moreover, we speculate that learning a world model can benefit model-free RL in this study. In our case, the world model is learned but not used for planning [159] or dreaming [82]. Instead, learning the world model can be considered an auxiliary task to facilitate downstream model-free RL by promoting representation learning of the shared neural network. One advantage of this scheme is that the agent can achieve reasonably well performance with an imperfect model, which is usually the case in human brains (for example, a professional game player does not need to know every detail of the game mechanisms). It is also reasonable to conjecture that oppositely, model-free RL, as an auxiliary objective, may facilitate learning the world model by sharing some neural modules.

However, the studies about the interplay between model-based and model-free RL were under-explored in both machine learning and neuroscience. Should learning the world model serve as an auxiliary task like in our and the related studies [102, 125]? Are there any other ways model-free and model-based RL benefit each other? It remains to be investigated more comprehensively in future neuroscience and machine learning research.

4.7.2 Representation learning and RL

In this work, we suggested one way to explicitly address the representation learning problem in deep RL. We focused on POMDP problems where historical observations and actions were useful. We trained an RNN world model to perform observation prediction so that the internal states of the RNN encoded important underlying environmental states that affected state transitions. Similar ideas for representation learning in POMDPs were also addressed by Igl et al. [102] and Lee et al. [125], while they were implemented in very different ways (see Chap. 4.2).

Note that these two studies [102, 125] and our work are intuitively effective in history-dependent POMDPs because extracting critical information from sequential data is non-trivial and essential. Meanwhile, Ota et al. [161][5] suggested that explicitly addressing the representation problem is even beneficial in MDPs. They also asked the model to predict the next state (but with feedforward networks) to facilitate representation learning [161]. This improved RL performance with different base algorithms in robotic control tasks.

Recently, there has also been rising research interest in using contrastive learning [32, 81] to perform representation for RL [123, 206][6]. Contrastive learning is a kind of self-supervised machine learning to learn the general features of a dataset (which data points are similar or different) without labels. One major advantage of contrastive learning is that reconstructing or predicting observations is not required. Thus, contrastive learning is especially computationally efficient for high-dimensional observations such as images. For example, Laskin et al. [123] used contrastive learning to extract features from image observations and performed off-policy RL based on the extracted features. Their experimental results showed a significant performance gain compared to RL without representation learning.

The aforementioned and our studies suggested that properly handling representation learning is critical for RL performance. However, how to "properly" address the representation learning problem in deep RL remains under-explored. A potential issue is the representation shift problem. We suggested a heuristic approach for alleviating the representation shift problem by employing two VRMs (first-impression and keep-leaning VRMs). However, it is still worthwhile to develop theoretically justified approaches to resolve the representation shift issue. Future studies may need more theoretical and empirical discussions on effective and robust representation learning for deep RL.

[5] Ota et al.'s paper [161] was published later than this work.

[6] These studies [123, 206] were also published later than this work.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Traditionally, machine learning has been most useful for processing high-dimensional data that are difficult for humans to comprehend, such as dimension reduction [69] and clustering [137]. However, the recent development of deep learning using neural networks (and the corresponding computational hardware) makes it possible for the machine learning agent to tackle more diverse tasks that humans also meet in real life. On the one hand, deep learning demonstrates supreme performance on some well-defined tasks such as image recognition [93] and machine translation [231]. On the other hand, more and more interest is shown in leveraging artificial neural networks to study the cognitive basis of decision-making [134, 145, 167, 202, 217, 238, 248, 250]. Nonetheless, current deep learning agents are still far behind humans in flexibility and versatility—the proposed studies in this thesis attempt to fill such gaps.

In the previous chapters, two studies of RL with stochastic RNN applied in hierarchical and partially observable environments were introduced. Here I shortly summarize these studies and discuss their potential extension for cognitive neurorobotics and decision-making research.

The first work introduces a bio-inspired RL framework, referred to as ReMASTER, for self-organizing action hierarchy with exploratory learning. The intrinsic timescale hierarchy of a two-layers RNN and neuronal stochasticity are the two key elements. Unlike other hierarchical RL methods, ReMASTER does not require designing and optimizing a specific objective for abstracting sub-goals (with the experimenter's prior knowledge). Instead, a representation of the action hierarchy emerged in the network only through RL. The self-organized action hierarchy is shown to help accelerate transfer learning with the learned low-level control skills. Moreover, the network can output the low-level control for an explicit sub-goal by clamping the higher-level neural activities. Our study demonstrates the potential of bio-inspired learning mechanisms for deep RL. In particular, the hierarchy of intrinsic timescales of the model and stochastic system dynamics are shown to be critical to the self-organization of reusable motor skills.

The second work focuses on RL in partially observable environments. The core argument is that in partially observable tasks, it could be beneficial to separately handle representation learning (i.e., learning to extract critical information from a

sequence of raw observations) and RL (i.e., learning the value and policy functions). The experimental results support this argument by using a variational RNN as the model for representation learning and acquiring the belief state in POMDPs. In human development, it is considered that a world model develops in much earlier stages via representation learning, such as predictive coding [170], than decision-making [100, 229]. Therefore, when an adult needs to learn a new decision-making task that requires historical or contextual observations, the person can immediately utilize the belief state by the learned model of the world. Since we deal with many different tasks in the same world that follow the same physical laws. The reusability of the representation model and belief state should contribute to a more adaptive decision-making capacity. In particular, finding the structure in time by learning to predict future observations should be helpful since optimal decisions are usually context-dependent in practice.

Both works demonstrated the advantages of using stochastic RNNs for RL from different perspectives. The stochasticity in the networks can contribute to more efficient exploration and better latent representations. The recurrent property of the network enables the model to keep the internal states which empower representations that are useful for hierarchical control and inferring the unobservable environmental state. As a short, final summary, the thesis studies suggest the potential of using RL with stochastic RNNs to advance more intelligent decision-making and motor control.

5.2 Future work

Despite the recent success of decision-making AI on various individual tasks [40, 197, 234], decision-making agents are still far from achieving versatility, adaptivity, and flexibility similar to or surpassing human beings. The possible future directions are shortly discussed as follows. Potential future work following the thesis studies may be conducted in several directions.

One ongoing work of the thesis author [89] considers the cooperation between RL and another decision-making paradigm: the active inference (AIf) theory [67, 68], which is an extension of the free energy principle (FEP) [65]. Aif and FEP provide a Bayesian computational framework of the decision-making in the brain. AIf explains decision-making as changing the agent's belief of perception and proprioception to minimize the difference (or *surprise* in FEP) between predicted observation and goal observation [66, 143, 216]. In this work, We consider a novel Bayesian framework for explaining habitual and goal-directed behaviors in decision making, in which everything is integrated as one variational neural network model. The model learns to predict future observations using the data collected by self-exploration and to generate motor actions by sampling stochastic internal states $\boldsymbol{z}$. Both the prior and posterior distributions of the Bayesian variable $\boldsymbol{z}$ are modeled by the network. One the one hand, habitual behavior, which is obtained from the prior distribution of $\boldsymbol{z}$, is acquired by RL. On the other hand, Goal-directed behavior is determined from the posterior distribution of $\boldsymbol{z}$ by using active inference, which computes $\boldsymbol{z}$ by minimizing the variational surprise between the predicted and desired future observation.

The second is to develop novel principles/mechanisms for more intelligent decision-making by borrowing ideas from biology and cognitive science. For example, there is

a *primacy bias* effect [141, 194] in cognitive science, a phenomenon in which human learning overly relies on the early experiences of a task. Reminded by this phenomenon, Nikishin et al. [157] proposed a simple yet general-applicable method to reduce the risk of over-fitting to earlier experiences in deep RL, which showed consistent performance improvement in various tasks. Another example is that Kirkpartrick et al. [116] developed an approach to overcome the *catastrophic forgetting* problem in deep learning (an ANN fails to maintain expertise on old tasks when learning new tasks [64]), inspired by synaptic consolidation in neuroscience. As for my future studies, an interesting but under-explored research topic is how the brain mechanisms during sleep can give insights into deep learning, as sleep is considered to play a critical role in learning [139, 237].

Moreover, I would like to improve the versatility of decision-making AIs (including but not limited to the proposed models in the thesis) to tackle more diverse, real-world problems. In particular, a deep RL model usually requires hyperparameter tuning to perform well in a given task (set). By contrast, a healthy human can learn various tasks with reasonably good performance. How a single decision-making AI model can effectively learn to solve various tasks without much hyperparameter tuning? The answers to this question should also be important to understand the versatility of our cognitive ability. One way to alleviate the demands of hyperparameter tuning is to introduce new hyperparameters to regularize the original hyperparameters (e.g., introducing the target entropy [84] to regularize the temperature of the policy function [83]). It is essential that there should exist a choice of the new hyperparameters that are good for a wide range of tasks. This approach may also apply to ReMASTER, where we can introduce mechanisms to adaptively adjust some critical hyperparameters. For instance, adaptive control of the scale of neuronal noise in ReMASTER should be beneficial, which may be implemented using variational RNNs like in Chap. 4. It will also be interesting to consider a compromise between totally-fixed and totally-adaptive intrinsic timescales of the RNN since they both have disadvantages. RNNs with totally-fixed timescales (like MTSRNN) may suffer from a disparity between the network timescales and task timescales, and RNNs with totally-adaptive intrinsic timescales (like LSTM) are shown to underperform the MTSRNN in the transfer learning tasks (Chap. 3). An interesting mechanism that may apply here is to use another RNN with slowly-changing RNN outputs, such as a CTRNN (Chap. 2.2.2), to regularize the timescale of the main RNN, which is proposed by Tay et al. [219].

Last but not least, it is worthwhile to consider more general principles (meta-mechanisms) that underlie a genuinely versatile and adaptive decision-making agent in the further future. While detailed artificial designs are essential in engineering problems for better performance, too many artificial designs with humans' prior knowledge about specific tasks are unnecessary and perhaps deleterious to developing and understanding a highly cognitive and intelligent decision-making agent. The animals in nature improve themselves with three levels of adaptation schemes—learning, development, and evolution. At the learning level, animals use life experience to improve skills with a developed body and brain, with is analog to optimizing a given neural network in deep learning. At the development level, animals grow their brains and bodies with progress pre-defined in the genes, which is analog to augmenting or changing the networks for life-long learning in deep learning [178]. At the evolution level, random

mutations occur, and better genes are selected, which is analog to neural architecture search in deep learning [256]. However, there have not been comprehensive studies that include all the three levels of adaptation in machine learning for decision-making problems, probably because of the immense demand for computation power and the fact that the learning level has not been perfectly addressed yet. In the further future, it will be interesting to study the development and evolution levels of deep RL, which update not only the networks but also the learning principles/algorithms of the agents.

Bibliography

[1] David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 20–29, 2018.

[2] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[3] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[4] Ahmadreza Ahmadi and Jun Tani. Bridging the gap between probabilistic and deterministic models: a simulation study on a variational Bayes predictive coding recurrent neural network model. In *International Conference on Neural Information Processing*, pages 760–769. Springer, 2017.

[5] Ahmadreza Ahmadi and Jun Tani. A novel predictive-coding-inspired variational RNN model for online prediction and recognition. *Neural Computation*, pages 1–50, 2019.

[6] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *Proceedings of the International Conference on Learning Representations*, 2018.

[7] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations*, 2017.

[8] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

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

[10] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] Akhil Bagaria and George Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020.

[12] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.

[13] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 834–846, 1983.

[14] Jeffrey M Beck, Wei Ji Ma, Roozbeh Kiani, Tim Hanks, Anne K Churchland, Jamie Roitman, Michael N Shadlen, Peter E Latham, and Alexandre Pouget. Probabilistic population codes for Bayesian decision making. *Neuron*, 60(6):1142–1152, 2008.

[15] Jeffrey M Beck, Wei Ji Ma, Xaq Pitkow, Peter E Latham, and Alexandre Pouget. Not noisy, just wrong: the role of suboptimal inference in behavioral variability. *Neuron*, 74(1):30–39, 2012.

[16] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[17] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[18] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[19] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[20] Anthony Boemio, Stephen Fromm, Allen Braun, and David Poeppel. Hierarchical and asymmetric temporal sensitivity in human auditory cortices. *Nature Neuroscience*, 8(3):389–395, 2005.

[21] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[22] Oliver Bown and Sebastian Lexer. Continuous-time recurrent neural networks for generative and interactive musical performance. In *Workshops on Applications of Evolutionary Computation*, pages 652–663. Springer, 2006.

[23] Jonathan L Brigman, Poonam Mathur, Judith Harvey-White, Alicia Izquierdo, Lisa M Saksida, Timothy J Bussey, Stephanie Fox, Evan Deneris, Dennis L Murphy, and Andrew Holmes. Pharmacological or genetic inactivation of the serotonin transporter improves reversal learning in mice. *Cerebral Cortex*, 20(8):1955–1963, 2010.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[25] Rishidev Chaudhuri, Kenneth Knoblauch, Marie-Alice Gariel, Henry Kennedy, and Xiao-Jing Wang. A large-scale circuit mechanism for hierarchical dynamical processing in the primate cortex. *Neuron*, 88(2):419–431, 2015.

[26] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[27] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the International Conference on Neural Information Processing Systems*, pages 6572–6583, 2018.

[28] Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double Q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

[29] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[30] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[31] Minkyu Choi and Jun Tani. Predictive coding for dynamic vision: Development of functional hierarchy in a multiple spatio-temporal scales RNN model. In *International Joint Conference on Neural Networks (IJCNN)*, pages 657–664. IEEE, 2017.

[32] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[33] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[34] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, pages 2980–2988, 2015.

[35] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[36] Albert Compte, Nicolas Brunel, Patricia S Goldman-Rakic, and Xiao-Jing Wang. Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cerebral Cortex*, 10(9):910–923, 2000.

[37] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[38] Ian Danforth. Continuous cartpole for OpenAI Gym. `https://gist.github.com/iandanforth/e3ffb67cf3623153e968f2afdfb01dc8`, 2018.

[39] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 5, 1992.

[40] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of Tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[41] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. In *Proceedings of the International Conference on Machine Learning*, pages 179–186, 2012.

[42] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning*, pages 465–472, 2011.

[43] Markus Diesmann, Marc-Oliver Gewaltig, and Ad Aertsen. Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402(6761):529–533, 1999.

[44] Thomas Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 12, 1999.

[45] Thomas G Dietterich. Hierarchical reinforcement learning with the maxQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[46] Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[47] Kenji Doya, Shin Ishii, Alexandre Pouget, and Rajesh PN Rao. *Bayesian brain: Probabilistic approaches to neural coding*. MIT press, 2007.

[48] Kenji Doya, Kayoko W Miyazaki, and Katsuhiko Miyazaki. Serotonergic modulation of cognitive computations. *Current Opinion in Behavioral Sciences*, 38:116–123, 2021.

[49] Kenji Doya and Tadahiro Taniguchi. Toward evolutionary and developmental intelligence. *Current Opinion in Behavioral Sciences*, 29:91–96, 2019.

[50] Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[51] Kazuki Enomoto, Naoyuki Matsumoto, Sadamu Nakai, Takemasa Satoh, Tatsuo K Sato, Yasumasa Ueda, Hitoshi Inokawa, Masahiko Haruno, and Minoru Kimura. Dopamine neurons learn to encode the long-term value of multiple future rewards. *Proceedings of the National Academy of Sciences*, page 201014457, 2011.

[52] Manfred Eppe, Christian Gumbsch, Matthias Kerzel, Phuong DH Nguyen, Martin V Butz, and Stefan Wermter. Intelligent problem-solving as integrated hierarchical reinforcement learning. *Nature Machine Intelligence*, pages 1–10, 2022.

[53] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[54] Daniel J Felleman and David C Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex (New York, NY: 1991)*, 1(1):1–47, 1991.

[55] Matthew Fellows, Anuj Mahajan, Tim GJ Rudner, and Shimon Whiteson. Virel: A variational inference framework for reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[56] David Ferrier. *The functions of the brain.* Smith, Elder, 1886.

[57] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*, pages 1126–1135, 2017.

[58] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[59] Shelly B Flagel, Jeremy J Clark, Terry E Robinson, Leah Mayo, Alayna Czuj, Ingo Willuhn, Christina A Akers, Sarah M Clinton, Paul EM Phillips, and Huda Akil. A selective role for dopamine in stimulus–reward learning. *Nature*, 469(7328):53–57, 2011.

[60] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[61] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. In *Proceedings of the International Conference on Learning Representations*, 2018.

[62] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pages 2199–2207, 2016.

[63] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *Proceedings of the International Conference on Learning Representations*, 2018.

[64] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[65] Karl Friston. The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010.

[66] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, Giovanni Pezzulo, et al. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016.

[67] Karl Friston, Jérémie Mattout, and James Kilner. Action understanding and active inference. *Biological Cybernetics*, 104(1):137–160, 2011.

[68] Karl J Friston, Jean Daunizeau, James Kilner, and Stefan J Kiebel. Action and behavior: a free-energy formulation. *Biological Cybernetics*, 102(3):227–260, 2010.

[69] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[70] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[71] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993.

[72] Thomas Furmston and David Barber. Variational methods for reinforcement learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 241–248. JMLR Workshop and Conference Proceedings, 2010.

[73] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving pilco with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, 2016.

[74] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. Multi-goal reinforcement learning environments for simulated franka emika panda robot. *arXiv preprint arXiv:2106.13687*, 2021.

[75] Karunesh Ganguly and Mu-ming Poo. Activity-dependent neural plasticity from bench to bedside. *Neuron*, 80(3):729–741, 2013.

[76] Jan Gläscher, Nathaniel Daw, Peter Dayan, and John P O'Doherty. States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595, 2010.

[77] Gary H Glover. Overview of functional magnetic resonance imaging. *Neurosurgery Clinics*, 22(2):133–139, 2011.

[78] Anirudh Goyal Alias Parth Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in Neural Information Processing Systems*, pages 6713–6723, 2017.

[79] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[80] Stephen Grossberg. Recurrent neural networks. *Scholarpedia*, 8(2):1888, 2013.

[81] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[82] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2450–2462. Curran Associates, Inc., 2018.

[83] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

[84] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[85] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[86] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[87] Dongqi Han, Kenji Doya, and Jun Tani. Self-organization of action hierarchy and compositionality by reinforcement learning with recurrent neural networks. *Neural Networks*, 129:149–162, 2020.

[88] Dongqi Han, Kenji Doya, and Jun Tani. Variational recurrent models for solving partially observable control tasks. In *International Conference on Learning Representations*, 2020.

[89] Dongqi Han, Kenji Doya, and Jun Tani. Goal-directed planning by reinforcement learning and active inference. *arXiv preprint arXiv:2106.09938*, 2021.

[90] Dongqi Han, Tadashi Kozuno, Xufang Luo, Zhao-Yun Chen, Kenji Doya, Yuqing Yang, and Dongsheng Li. Variational oracle guiding for reinforcement learning. In *International Conference on Learning Representations*, 2022.

[91] Christoph Hartmann, Andreea Lazar, Bernhard Nessler, and Jochen Triesch. 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, 2015.

[92] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015.

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[94] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[95] Claus-C Hilgetag, Marc A O'Neill, and Malcolm P Young. Hierarchical organization of macaque and cat cortical sensory systems explored with a novel network processor. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 355(1393):71–89, 2000.

[96] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[97] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[98] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[99] Ronald A Howard. *Dynamic programming and Markov processes*. Wiley for The Massachusetts Institute of Technology, 1964.

[100] Elizabeth B Hurlock. *Developmental psychology.* McGraw-Hill, 1953.

[101] Raoul Huys, Andreas Daffertshofer, and Peter J Beek. Multiple time scales and multiform dynamics in learning to juggle. *Motor Control*, 8(2):188–212, 2004.

[102] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. *arXiv preprint arXiv:1806.02426*, 2018.

[103] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[104] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[105] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in Psychology*, volume 121, pages 471–495. Elsevier, 1997.

[106] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[107] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[108] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[109] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for Atari. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[110] Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2018.

[111] Nan Rosemary Ke, Amanpreet Singh, Ahmed Touati, Anirudh Goyal, Yoshua Bengio, Devi Parikh, and Dhruv Batra. Learning dynamics model in reinforcement learning by incorporating the long term future. *arXiv preprint arXiv:1903.01599*, 2019.

[112] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[113] IS Khalil, JC Doyle, and K Glover. *Robust and optimal control*. prentice hall, new jersey, 1996.

[114] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[115] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[116] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[117] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems*, 22, 2009.

[118] Konrad P Kording, Joshua B Tenenbaum, and Reza Shadmehr. The dynamics of memory as a consequence of optimal adaptation to a changing body. *Nature Neuroscience*, 10(6):779, 2007.

[119] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[120] Rudolf Kruse, Christian Borgelt, Frank Klawonn, Christian Moewes, Matthias Steinbrecher, and Pascal Held. Multi-layer perceptrons. In *Computational Intelligence*, pages 47–81. Springer, 2013.

[121] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

[122] Zeb Kurth-Nelson and A David Redish. Temporal-difference reinforcement learning with distributed representations. *PLoS One*, 4(10):e7362, 2009.

[123] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[124] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[125] Alex Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[126] Sang Wan Lee, Shinsuke Shimojo, and John P O'Doherty. Neural computations underlying arbitration between model-based and model-free learning. *Neuron*, 81(3):687–699, 2014.

[127] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[128] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review, 2018.

[129] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of International Conference on Learning Representations*, 2019.

[130] Alexander Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2020.

[131] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering Mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

[132] Zhuoru Li, Akshay Narayan, and Tze-Yun Leong. An efficient approach to model-based hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[133] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[134] Timothy P Lillicrap and Konrad P Kording. What does it mean to understand a neural network? *arXiv preprint arXiv:1907.06374*, 2019.

[135] Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current Opinion in Neurobiology*, 55:82–89, 2019.

[136] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[137] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[138] Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Adaptive skills adaptive partitions (asap). In *Advances in Neural Information Processing Systems*, pages 1588–1596, 2016.

[139] Pierre Maquet. The role of sleep in learning and memory. *Science*, 294(5544):1048–1052, 2001.

[140] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-timing-dependent plasticity: a comprehensive overview. *Frontiers in Synaptic Neuroscience*, 4:2, 2012.

[141] Philip H Marshall and Pamela R Werder. The effects of the elimination of rehearsal on primacy and recency. *Journal of Verbal Learning and Verbal Behavior*, 11(5):649–653, 1972.

[142] Sara Matias, Eran Lottem, Guillaume P Dugué, and Zachary F Mainen. Activity patterns of serotonin neurons underlying cognitive flexibility. *Elife*, 6:e20552, 2017.

[143] Takazumi Matsumoto and Jun Tani. Goal-directed planning for habituated agents by active inference using a variational recurrent neural network. *Entropy*, 22(5):564, 2020.

[144] R Andrew McCallum. Overcoming incomplete perception with utile distinction memory. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 190–196, 1993.

[145] Kevin R McKee, Edward Hughes, Tina O Zhu, Martin J Chadwick, Raphael Koster, Antonio Garcia Castaneda, Charlie Beattie, Thore Graepel, Matt Botvinick, and Joel Z Leibo. Deep reinforcement learning models the emergent dynamics of human cooperation. *arXiv preprint arXiv:2103.04982*, 2021.

[146] Jacques Mirenowicz and Wolfram Schultz. Importance of unpredictability for reward responses in primate dopamine neurons. *Journal of Neurophysiology*, 72(2):1024–1027, 1994.

[147] Kayoko W Miyazaki, Katsuhiko Miyazaki, Kenji F Tanaka, Akihiro Yamanaka, Aki Takahashi, Sawako Tabuchi, and Kenji Doya. Optogenetic activation of dorsal raphe serotonin neurons enhances patience for future rewards. *Current Biology*, 24(17):2033–2040, 2014.

[148] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[149] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[150] Karin Morandell and Daniel Huber. The role of forelimb motor cortex areas in goal directed action in mice. *Scientific Reports*, 7(1):15759, 2017.

[151] Jun Morimoto and Kenji Doya. Robust reinforcement learning. *Neural Computation*, 17(2):335–359, 2005.

[152] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender. Complexity of finite-horizon Markov decision process problems. *Journal of the ACM (JACM)*, 47(4):681–720, 2000.

[153] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[154] John D Murray, Alberto Bernacchia, David J Freedman, Ranulfo Romo, Jonathan D Wallis, Xinying Cai, Camillo Padoa-Schioppa, Tatiana Pasternak, Hyojung Seo, Daeyeol Lee, et al. A hierarchy of intrinsic timescales across primate cortex. *Nature Neuroscience*, 17(12):1661, 2014.

[155] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[156] Karl M Newell, Yeou-Teh Liu, and Gottfried Mayer-Kress. Time scales in motor learning and development. *Psychological Review*, 108(1):57, 2001.

[157] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2022.

[158] John P O'Doherty, Peter Dayan, Karl Friston, Hugo Critchley, and Raymond J Dolan. Temporal difference models and reward-related learning in the human brain. *Neuron*, 38(2):329–337, 2003.

[159] Masashi Okada, Norio Kosaka, and Tadahiro Taniguchi. Planet of the Bayesians: Reconsidering and improving deep planning network by incorporating Bayesian inference. In *International Conference on Intelligent Robots and Systems*, 2020.

[160] Gergő Orbán, Pietro Berkes, József Fiser, and Máté Lengyel. Neural variability and sampling-based probabilistic representations in the visual cortex. *Neuron*, 92(2):530–543, 2016.

[161] Kei Ota, Tomoaki Oiki, Devesh Jha, Toshisada Mariyama, and Daniel Nikovski. Can increasing input dimensionality improve deep reinforcement learning? In *International Conference on Machine Learning*, pages 7424–7433. PMLR, 2020.

[162] Christos H Papadimitriou and John N Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[163] Gibeom Park and Jun Tani. Development of compositional and contextual communication of robots by using the multiple timescales dynamic neural network. In *Development and Learning and Epigenetic Robotics (ICDL-EpiRob), 2015 Joint IEEE International Conference on*, pages 176–181. IEEE, 2015.

[164] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.

[165] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10, 1997.

[166] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[167] Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *International Conference on Machine Learning*, pages 4218–4227. PMLR, 2018.

[168] Pasko Rakic. Evolution of the neocortex: a perspective from developmental biology. *Nature Reviews Neuroscience*, 10(10):724–735, 2009.

[169] Steve Ramirez, Xu Liu, Pei-Ann Lin, Junghyup Suh, Michele Pignatelli, Roger L Redondo, Tomás J Ryan, and Susumu Tonegawa. Creating a false memory in the hippocampus. *Science*, 341(6144):387–391, 2013.

[170] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79, 1999.

[171] Stéphanie Ratté, Sungho Hong, Erik De Schutter, and Steven A Prescott. Impact of neuronal properties on network coding: roles of spike initiation dynamics and robust synchrony transfer. *Neuron*, 78(5):758–772, 2013.

[172] Chris Reinke. *The Gamma-Ensemble-Adaptive Reinforcement Learning via Modular Discounting*. PhD thesis, Okinawa Institute of Science and Technology Graduate University, 2018.

[173] Chris Reinke, Eiji Uchibe, and Kenji Doya. Average reward optimization with multiple discounting reinforcement learners. In *International Conference on Neural Information Processing*, pages 789–800. Springer, 2017.

[174] Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In *Advances in Neural Information Processing Systems*, pages 10424–10434, 2018.

[175] Robert D Rogers, BJ Everitt, A Baldacchino, Alison J Blackshaw, Rachel Swainson, K Wynne, NB Baker, J Hunter, T Carthy, E Booker, et al. Dissociable

deficits in the decision-making cognition of chronic amphetamine abusers, opiate abusers, patients with focal damage to prefrontal cortex, and tryptophan-depleted normal volunteers: evidence for monoaminergic mechanisms. *Neuropsychopharmacology*, 20(4):322–339, 1999.

[176] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[177] Caroline A Runyan, Eugenio Piasini, Stefano Panzeri, and Christopher D Harvey. Distinct timescales of population coding across cortex. *Nature*, 548(7665):92, 2017.

[178] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[179] Katsuyuki Sakai, Katsuya Kitaguchi, and Okihide Hikosaka. Chunking during human visuomotor sequence learning. *Experimental Brain Research*, 152(2):229–242, 2003.

[180] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320. PMLR, 2015.

[181] J Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. *Institut für Informatik, Technische Universität München. Technical Report FKI-126*, 90, 1990.

[182] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning*. Diploma thesis, Technische Universität München, 1987.

[183] Jürgen Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In *Advances in Neural Information Processing Systems*, pages 500–506, 1991.

[184] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[185] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, chess and Shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[186] Nicolas Schweighofer, Mathieu Bertin, Kazuhiro Shishida, Yasumasa Okamoto, Saori C Tanaka, Shigeto Yamawaki, and Kenji Doya. Low-serotonin levels increase delayed reward discounting in humans. *Journal of Neuroscience*, 28(17):4528–4532, 2008.

[187] Nicolas Schweighofer, Saori C Tanaka, and Kenji Doya. Serotonin and the evaluation of future rewards: theory, experiments, and possible neural mechanisms. *Annals of the New York Academy of Sciences*, 1104(1):289–300, 2007.

[188] Terrence J Sejnowski, Christof Koch, and Patricia Smith Churchland. Computational neuroscience. *Science*, 241(4871):1299–1306, 1988.

[189] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2020.

[190] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

[191] Katsunari Shibata and Yuta Sakashita. Reinforcement learning with internal-dynamics-based exploration using a chaotic neural network. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[192] Keisetsu Shima and Jun Tanji. Both supplementary and presupplementary motor areas are crucial for the temporal organization of multiple movements. *Journal of Neurophysiology*, 80(6):3247–3260, 1998.

[193] Keisetsu Shima and Jun Tanji. Neuronal activity in the supplementary and presupplementary motor areas for temporal organization of multiple movements. *Journal of Neurophysiology*, 84(4):2148–2160, 2000.

[194] Hanan Shteingart, Tal Neiman, and Yonatan Loewenstein. The role of first impression in operant learning. *Journal of Experimental Psychology: General*, 142(2):476, 2013.

[195] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, volume 13, page 05, 2013.

[196] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[197] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017.

[198] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems*, 23, 2010.

[199] Maurice A Smith, Ali Ghazizadeh, and Reza Shadmehr. Interacting adaptive processes with different timescales underlie short-term motor learning. *PLoS Biology*, 4(6):e179, 2006.

[200] Peter Smittenaar, Thomas HB FitzGerald, Vincenzo Romei, Nicholas D Wright, and Raymond J Dolan. Disruption of dorsolateral prefrontal cortex decreases model-based in favor of model-free control in humans. *Neuron*, 80(4):914–919, 2013.

[201] William R Softky and Christof Koch. The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps. *Journal of Neuroscience*, 13(1):334–350, 1993.

[202] H Francis Song, Guangyu R Yang, and Xiao-Jing Wang. Reward-based training of recurrent neural networks for cognitive and value-based tasks. *Elife*, 6:e21492, 2017.

[203] Yuhang Song, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Can the brain do backpropagation?—exact implementation of backpropagation in predictive coding networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 22566–22579, 2020.

[204] Mario Srouji, Jian Zhang, and Ruslan Salakhutdinov. Structured control nets for deep reinforcement learning. In *International Conference on Machine Learning*, pages 4742–4751. PMLR, 2018.

[205] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature Neuroscience*, 20(11):1643, 2017.

[206] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR, 2021.

[207] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[208] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[209] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

[210] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[211] Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

[212] Aboozar Taherkhani, Ammar Belatreche, Yuhua Li, Georgina Cosma, Liam P Maguire, and T Martin McGinnity. A review of learning in biologically plausible spiking neural networks. *Neural Networks*, 122:253–272, 2020.

[213] Keiji Tanaka. Neuronal mechanisms of object recognition. *Science*, 262(5134):685–688, 1993.

[214] Saori C Tanaka, Kenji Doya, Go Okada, Kazutaka Ueda, Yasumasa Okamoto, and Shigeto Yamawaki. Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. *Nature Neuroscience*, 7(8):887–893, 2004.

[215] Saori C Tanaka, Kenji Doya, Go Okada, Kazutaka Ueda, Yasumasa Okamoto, and Shigeto Yamawaki. Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. In *Behavioral economics of preferences, choices, and happiness*, pages 593–616. Springer, 2016.

[216] Jun Tani. Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(3):421–436, 1996.

[217] Jun Tani. *Exploring robotic minds: actions, symbols, and consciousness as self-organizing dynamic phenomena*. Oxford University Press, 2016.

[218] Jun Tanji and Keisetsu Shima. Role for supplementary motor area cells in planning several movements ahead. *Nature*, 371(6496):413–416, 1994.

[219] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. Recurrently controlled recurrent networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[220] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[221] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[222] Kurt A Thoroughman and Reza Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407(6805):742–747, 2000.

[223] Sebastian Thrun. Monte Carlo POMDPs. *Advances in Neural Information Processing Systems*, 12, 1999.

[224] Sebastian Thrun and Tom M Mitchell. Learning one more thing. Technical report, Carnegie-Mellon Univ Pittsburgh PA Dept of computer science, 1994.

[225] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[226] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[227] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical Review*, 36(5):823, 1930.

[228] Hiroki Utsunomiya and Katsunari Shibata. 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*, pages 970–978. Springer, 2008.

[229] Jeroen JA Van Boxtel and Hongjing Lu. A predictive coding perspective on autism spectrum disorders. *Frontiers in Psychology*, 4:19, 2013.

[230] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(11), 2008.

[231] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[232] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 3540–3549. JMLR. org, 2017.

[233] Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. Leverage the average: an analysis of regularization in rl. *arXiv preprint arXiv:2003.14089*, 2020.

[234] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[235] Nikos Vlassis, Michael L Littman, and David Barber. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Transactions on Computation Theory (TOCT)*, 4(4):1–8, 2012.

[236] Eric T Vu, Mark E Mazurek, and Yu-Chien Kuo. Identification of a forebrain motor programming network for the learned song of zebra finches. *Journal of Neuroscience*, 14(11):6924–6934, 1994.

[237] Gordon Wang, Brian Grone, Damien Colas, Lior Appelbaum, and Philippe Mourrain. Synaptic plasticity in sleep: learning, homeostasis and disease. *Trends in Neurosciences*, 34(9):452–463, 2011.

[238] Jane X Wang, Zeb Kurth-Nelson, Dharshan Kumaran, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Demis Hassabis, and Matthew Botvinick. Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, 21(6):860, 2018.

[239] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[240] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *Proceedings of the International Conference on Learning Representations*, 2017.

[241] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[242] Theophane Weber, Nicolas Heess, Ali Eslami, John Schulman, David Wingate, and David Silver. Reinforced variational inference. In *Advances in Neural Information Processing Systems (NIPS) Workshops*, 2015.

[243] James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019.

[244] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer, 2007.

[245] Roy A Wise. Dopamine, learning and motivation. *Nature Reviews Neuroscience*, 5(6):483–494, 2004.

[246] Bohan Wu, Jayesh K Gupta, and Mykel Kochenderfer. Model primitives for hierarchical lifelong reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–38, 2020.

[247] Kelvin Xu, Siddharth Verma, Chelsea Finn, and Sergey Levine. Continual learning of control primitives: Skill discovery via reset-games. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[248] Yuichi Yamashita and Jun Tani. Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment. *PLoS Computational Biology*, 4(11):e1000220, 2008.

[249] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, 19(3):356–365, 2016.

[250] Guangyu Robert Yang, Madhura R Joglekar, H Francis Song, William T Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 22(2):297, 2019.

[251] Zhaoyang Yang, Kathryn Merrick, Lianwen Jin, and Hussein A Abbass. Hierarchical deep reinforcement learning for continuous action control. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5174–5184, 2018.

[252] Haiyan Yin, Jianda Chen, Sinno Jialin Pan, and Sebastian Tschiatschek. Sequential generative exploration model for partially observable reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10700–10708, 2021.

[253] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[254] Andreas Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.

[255] Marvin Zhang, Zoe McCarthy, Chelsea Finn, Sergey Levine, and Pieter Abbeel. Learning deep neural network policies with continuous memory states. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 520–527. IEEE, 2016.

[256] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.