

SENSORIMOTOR ASPECTS OF BRAIN FUNCTION: DEVELOPMENT, INTERNAL
DYNAMICS, AND TOOL USE

A Dissertation

by

JAE WOOK YOO

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Chair of Committee, Yoonsuck Choe
Committee Members, Ricardo Gutierrez-Osuna
Dylan Shell
Pilwon Hur
Head of Department, Dilma Da Silva

May 2018

Major Subject: Computer Science

Copyright 2018 Jae Wook Yoo

ABSTRACT

Learning through the sensorimotor loop is essential for intelligent agents. While the important role of sensorimotor learning has been studied, several important aspects of sensorimotor learning in the brain, such as the development of motor behavior maps, the influence of internal dynamics on external behaviors, and the emergence of tool-use capabilities, are not addressed significantly. In this dissertation, I will address three questions trying to probe the nature of sensorimotor learning: (1) How can a sensorimotor agent understand its own body by developing a cortical map of its motor actions?; (2) How can predictive internal brain dynamics play an important role in external behavior (authorship of actions)?; (3) How tool-use emerges in a sensorimotor system interacting with the environment?

The first topic considers developing a cortical motor action map in a sensorimotor agent. Motivated by an experimental study showing a topographical map of complex behaviors in the macaque brain, we developed a target reaching gesture map using a biologically motivated self-organizing map model of the cortex with two-joint arm movements as inputs. The resulting gesture map showed a global topographic order based on the target locations. The map is comparable to the motor map reported in the experimental study.

In the second topic of this dissertation, I discuss and investigate the role of the predictive internal brain dynamics. Previous computer simulation studies showed that neural network controllers with more predictable internal state dynamics can attain higher performance in harsher or changing environments. This implies that predictable internal state dynamics could be a necessary condition for intelligent agents in the evolutionary pathway to have authorship of actions and to adapt themselves to changing environment. However, there was a missing link; the findings from the simulation study do not necessarily mean that the internal state dynamics affects the external behavior (authorship of actions) in the biological

brain as well. To fill the gap, I investigated the role of predictability of internal state dynamics in the brain by analyzing the human EEG data. These results support our hypothesis on the existence of predictable dynamics and its relation to conscious states (as a surrogate of authorship of actions).

Lastly, in the third topic of this dissertation, I present tool-use in sensorimotor agents. Tool-use requires high levels of sensorimotor skill learning and problem solving capabilities and is one of the salient indicators of intelligence along with communication (language) and logic. However, through our literature search, we found that there are two gaps in tool-use in AI and robotics. First, most works depend on some degree of designer knowledge regarding tool-use and motor control. Furthermore, tool-use tasks that require multiple subtasks to be completed in specific order are almost non-existent in deep reinforcement learning (RL) literature for developments and benchmarks, even though deep RL has demonstrated good performance in some control tasks in recent years. In this dissertation, I present environments and sensorimotor systems where the agents can adapt to use simple or complex tools based on minimal task knowledge. Specifically, I present two approaches. I first evolve neural network controllers for simple tool-use behavior in reaching tasks with minimal task knowledge, followed by analysis of the evolved networks. The results show that minimal, indirect fitness criteria are enough to give rise to tool-use behavior. Then, as a second step, I implement a more complex tool-use environment such as dragging an object to a target location using a tool in a physics simulation, and demonstrate a deep RL method with stepwise composite reward shaping methodology to learn the complex tool-use task successfully.

Overall, the studies in this dissertation are expected to help researchers in brain science and AI understand the nature of sensorimotor aspects of learning in the brain and implement them in AI.

DEDICATION

To my fiancée, my brother, and my parents.

ACKNOWLEDGEMENTS

Above all, I would like to express my deepest appreciation to my advisor Dr. Yoonsuck Choe for his guidance and support during my graduate studies. Whenever I reached research obstacles, he has guided me towards the right path and encouraged me to move forward to reach my goal. I could once again feel that I learned a lot from him while preparing my dissertation and defense. I have the greatest respect for his personality, intuition, and attitude as a scholar.

I would also like to thank my committee members Dr. Ricardo Gutierrez-Osuna, Dr. Dylan Shell, and Dr. Pilwon Hur for their support and valuable comments on improving my research.

During my studies at Texas A&M, I had the privilege to work with wonderful colleagues and professors. Dr. John Keyser, Dr. Louise Abbott, Dr. Jaerock Kwon, Dr. Hyun Chul Chung, Junseok Lee, Michael Nowak, Qinbo Li, Han Wang, Khuong Nguyen, Qing Wan, and Jinho Choi. I would like to thank them for their support and feedback.

Last but not least, I would like to thank my fiancée Moonjeong Kang, my brother Jae Suk Yoo, and my parents Moon Ok Yoo and Kyunghee Byune for their support, patience, and love.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr. Yoonsuck Choe, Dr. Ricardo Gutierrez-Osuna, and Dr. Dylan Shell of the Department of Computer Science and Engineering, and Dr. Pilwon Hur of the Department of Mechanical Engineering at Texas A&M University.

All work for the dissertation was completed by the student, in collaboration with Jinho Choi (for Chapter 2) and Qinbo Li (for Chapter 4) of the Department of Computer Science and Engineering, and Dr. Jaerock Kwon (for Chapter 3) of Electrical and Computer Engineering at Kettering University, under the advisement of Dr. Yoonsuck Choe of the Department of Computer Science and Engineering at Texas A&M University.

Funding Sources

Graduate study was partially supported by assistantships from the Department of Computer Science and Engineering at Texas A&M University.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
DEDICATION	iv
ACKNOWLEDGEMENTS	v
CONTRIBUTORS AND FUNDING SOURCES	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Approach	3
1.3 Organization of this Dissertation	3
2. DEVELOPMENT: SELF-ORGANIZATION OF THE MOTOR MAP IN THE CORTEX	5
2.1 Background	5
2.2 Related Work	7
2.3 The GCAL Model	8
2.4 Experiment	12
2.4.1 Experiment Platform	12
2.4.2 Generating movements	13
2.4.3 Experiment Procedure	13
2.5 Results	16
2.5.1 Local topography based on target location similarity	16
2.5.2 Global topographic order	16
2.6 Discussion	16
2.7 Contribution	19
3. INTERNAL DYNAMICS: PREDICTABLE INTERNAL BRAIN DYNAMICS AND ITS ROLE IN AUTHORSHIP OF ACTIONS	20
3.1 Introduction	20
3.2 Materials and Methods	24
3.2.1 EEG data	24
3.2.2 EEG data analysis	24

3.3	Results	28
3.4	Discussion	29
4.	EMERGENCE OF TOOL USE BY EVOLVED NEURAL CIRCUITS	35
4.1	Introduction	35
4.1.1	Tool Use in Animals	35
4.1.2	Tool Use in AI and Robotics	37
4.1.3	Approach	42
4.2	Methods	44
4.2.1	Algorithm for Evolving Neural Circuits	44
4.2.2	Fitness Criteria	47
4.2.3	Input and Output Interface for the Neural Circuit	48
4.3	Experiment (Simulation)	51
4.3.1	Reaching Task	51
4.3.2	Articulated Limb	52
4.3.3	Experimental (Simulation) Procedure	52
4.4	Results	53
4.4.1	Evolved Neural Circuits and Observed Behavior	53
4.4.2	Success Rate for Each Fitness Criterion	53
4.4.3	Contribution of Recurrent Connections	59
4.4.4	Internal Dynamics	61
4.5	Discussion	63
4.6	Conclusion	63
5.	END-TO-END TOOL USE LEARNING IN PHYSICS SIMULATION WITH DEEP REINFORCEMENT LEARNING	65
5.1	Introduction	65
5.2	Approach / Method	67
5.2.1	Tool-use Environment in a Physics Simulator	67
5.2.2	Stepwise Composite Reward Shaping	68
5.2.3	Deep RL Method: Actor-Critic Policy Gradient Method using Kronecker-Factored Trust Region	74
5.3	Experiments	81
5.4	Results and Analysis	83
5.4.1	Behavior analysis of the trained agents	84
5.4.2	Comparing performance of the trained agents	87
5.4.3	Learning curves	94
5.5	Discussion	100
5.6	Conclusion	103
6.	DISCUSSION AND CONCLUSION	105
	REFERENCES	109

LIST OF FIGURES

FIGURE	Page
<p>2.1 The topographic map found in precentral cortex of Monkey. (a) The enlarged view at the bottom shows the sites of the electrical microstimulation. The movements shown in the rectangles A–H were evoked by stimulating the sites A–H in the enlarged circle at the bottom. The stimulation of the right side of the brain caused mainly the left side of the body (left arm to move). (b) A shows the distribution of hand positions along the vertical axis, which are upper, middle, and lower space. After each stimulation, the evoked final target positions were used to categorize the site. B shows the distribution of hand positions along the horizontal axis, which are contralateral (right when using left hand), central, and ipsilateral (left when using left hand) space. Adapted from [46].</p>	6
<p>2.2 The GCAL architecture. In the model, the retina size was increased to 2.0 (side) to fit the input image size (80×80 pixels) and the projection area enlarged to 1.5 (radius) to project all parts of the arm movements. <i>Note that in the text we will use the GCAL terminology of retina, LGN, and V1 to refer to the sensory surface, thalamus, and cortex in our gesture map model, respectively.</i></p>	10
<p>2.3 The kinematics and movement of the two-joint arm. (a) The arm consists of two joint J1 (θ_1) : J2 (θ_2), and the arm L1 : L2 (with the length ratio 1.6 : 1). The θ_1 and θ_2 are randomly picked initially and change toward the target. (b) The arm movement is encoded as time-lapse image where time is encoded as the pixel intensity. The darkest one is the target (most recent) posture.</p>	14
<p>2.4 Examples of movements with 24 target locations. Starting from a random posture, move toward to one of 24 target locations (postures). The movement over time is expressed using different pixel intensity (darker = more recent). (a) Example movements with 16 distal target locations. (b) Example movements with 8 proximal target locations. These movements simulate the arm movements in the experimental literature (Fig.2.1a). Note: For the same target location, many different time-lapse images were generated by varying the initial posture.</p>	15

2.5	The resulting gesture maps of LGN OFF to V1 projection. 17×17 RFs are plotted from 48×48 cortex density to see the details of them. The enlarged views show the zoomed in views of 3×3 RFs at each corner. Note: LGN OFF and LGN ON patterns are exact inverses of each other and thus contain the same information. The learned projections to V1 from these two sheets were similar as a result, so here we only showed the LGN OFF to V1 projections which is easier to visually inspect.	17
2.6	(a) Target vectors estimated from the resulting gesture map receptive fields (Fig. 2.5). The direction and the length of each arrow show the target location's direction and the distance from the center. (b) The color maps of the horizontal and the vertical components of the vectors in (a): bright=high (right, up), dark=low (left, down). The color maps were convolved with a Gaussian filter of size 15×15 pixels and sigma 2.5 to show more clearly the global order.	18
3.1	Prediction of Internal state trajectory in recurrent neural network controllers for a pole-balancing task. (a) 2D Pole balancing task, where x and y are the coordinates of the cart, and θ_x and θ_y are the angles of the pole from the z-axis. (b) A recurrent neural network controller for the 2D pole balancing task (lower left). The hidden perceptrons' activations (internal state) over time (lower right). A 3D plot of the internal state trajectory (upper right). (c) Measuring predictability of the internal state trajectory, where measuring the prediction error on a next activation value ($t + 1$), given several past activation values as inputs ($t - 3$ to t) (d) The experimental design shows the population of recurrent neural network controllers (left), selecting high-perform controllers (middle), and post-selection analysis by means of Internal State Predictability (ISP) (right). Each controller that passed the selection state has equal task performance, but the analysis of ISP shows that some with high ISP and other with much less ISP. Interestingly, with a harder pole balancing task (by increasing the initial tilt angle of the pole), the controllers with high ISP mostly retain their performance, but those with low ISP lose most of their performance. Adapted from our recent work [69].	21
3.2	EEG data [62] from the PhysioBank [43] are shown. Each row represent data from each subject (four total) and each column represent different states. A. Awake, raw data. B. Awake, smoothed (Gaussian filter, $\sigma = 1$) and peaks identified (circles). C. REM, raw data. D. REM, smoothed and peaks identified. E. SWS, raw data. F. SWS, smoothed and peaks identified. Each data set had 30,000 data points but here we are showing only the first 1,000 for a better view of the details. Adapted from our recent work [151].	25

3.3	A neural network predictor for a time series. The network predict a value x at time $t + 1$ by looking k previous time steps from the current one at time t . The neurons (perceptrons) are interconnected, and the weights of the connections are adjusted to minimize the errors between the predicted output values and the ones in the train sets during training phase. We used $k=10$ past data points as inputs, 10 hidden units, and 1 output unit. The network was trained using the Levenberg-Marquardt algorithm, following [50]. In the algorithm, a damping parameter determines how much the algorithm will approximate Newton’s method (small value) or gradient descent (large value). The parameter was initially set to 0.001 and its decrease factor set to 0.1 and increase factor set to 10. Training was stopped when the validation set error failed to decrease for 5 consecutive epochs or if the gradient value fell below 10^{-10} . Note, however, that the particular type of algorithm used for the prediction is not of central importance and we expect similar results with any other reasonable algorithm.	27
3.4	Summary of EEG IPI Prediction Error Results (Mean and Standard Deviation). Mean and standard deviation of IPI prediction error are shown for all four subjects, for all three conditions (awake, REM, and SWS). The unit for the y -axis was 10 milliseconds. For all subjects, awake and REM conditions resulted in lower IPI prediction error than SWS, showing that predictive dynamics may be more prominent during conscious states. All differences were significant (t-test, $p < 10^{-6}$), except for REM vs. AWAKE for subject 4. See text for details. Awake state having higher IPI prediction error than REM state is somewhat unexpected, which we will discuss further in the discussion section. Adapted from our recent work [151].	30
3.5	EEG IPI Prediction Error Distribution The IPI prediction error distribution is shown for all four subjects, each for all three conditions (awake [red], REM [blue], and SWS [green]). The x -axis is in linear scale while the y -axis is in log scale for a clearer view of the probability of extreme error values. The unit for the x -axis was 10 milliseconds. The trends are consistent for all four subjects. REM has the highest peak near zero error, closely followed by awake state, and finally SWS which shows the lowest peak. SWS has the heaviest tail, meaning that high error values are much more common than awake state or REM. Adapted from our recent work [151].	30
3.6	EEG IPI Distribution A-D. The IPI distributions are shown for all four subjects, each for all three conditions (awake [red], REM [blue], and SWS [green]). For all cases, the IPI distributions are positively skewed. E. The skewness values are in the table.	33

4.1	Tool-body assimilation (Receptive Field Changes Due to Tool Use). The receptive fields for distal (end effector: hand) and proximal (first joint: shoulder) parts of the arm of bimodal (visual [pink] and somatosensory [blue]) neurons are shown. The visual RFs of these neurons are known to extend due to tool-use: (c) and (g). Adapted from [78].	38
4.2	Neuroevolution of Augmenting Topologies (NEAT). (a) The genotype-to-phenotype mapping in NEAT is shown. Each node and each connection has a gene. Each connection gene has an enable/disable flag and a unique identifier, the innovation number. (b) Crossover of two parents with different topology is shown. The genes of the two parents are aligned so that the innovation numbers match up. Adapted from [118].	46
4.3	Agent-centered sensory representation (AC) and kinematics of the joint arm. (a) AC uses a polar coordinate system. φ_1, φ_2 , and φ_3 represent the angles for the end effector, the target, and the tool, respectively. d_1, d_2 , and d_3 indicate the distances to the end effector, the target, and the tool, respectively. (b) The limb consists of two joints J1 (θ_1) : J2 (θ_2), and the arm segments $L1$: $L2$ (with the length ratio of $L1$: $L2 = 1$: 1.25). The two joint angles θ_1 : θ_2 are controlled by the neural circuit output.	49
4.4	Neural circuit controller and the jointed limb. (a) A neural circuit controller consists of eight sensory inputs, two motor outputs, and zero or more hidden neurons. The sensory inputs are RAC between the end effector and the target, RAC between the end effector and the tool, two joints angles (θ_1, θ_2), and two limit detectors for θ_1 and θ_2 . The motor outputs are for the two joint angles. The neurons are connected with excitatory (solid) and inhibitory connections (dashed) including recurrent connections. (b) The interaction cycle between the neural circuit controller and the environment is shown. When the end-effector reaches the tool end, the tool is automatically attached to the limb and the end of the tool becomes the end-effector. Note: the controller was not explicitly notified of the limb extension due to tool pick-up.	50
4.5	Example task conditions. The environment consists of two degree-of-freedom articulated limb, a target object, and a tool, all on a 2D plane. Note that the tool's initial locations are variable. The two half-circles indicate the original reach (inner) and the reach with tool-use (outer).	51

4.6	Examples of evolved neural circuits. (a) Evolved neural circuit with fitness criterion S^2T (speed squared and tool pick-up frequency): 24 neurons (8 sensory inputs, 2 motor outputs, and 14 hidden neurons) and 90 connections (49 excitatory and 41 inhibitory connections; 10 are recurrent). (b) Evolved neural circuit with fitness criterion DS (distance and speed): 45 neurons (8 sensory inputs, 2 motor outputs, and 35 hidden neurons) and 252 connections (124 excitatory and 128 inhibitory connections; 10 are recurrent).	54
4.7	Time-lapse images of representative limb movements. The three images in the top row ((a), (b), and (c)) show examples of successful movements (with fitness criteria S^2T). In the movements of (a) and (b), the limbs (blue and red lines) first move towards the tool (green horizontal line) to pick it up (i.e., the limb is extended), and then move towards the target (\square). In the movement of (c), the limb moves directly toward the target without picking up the tool. The trajectories of the end effectors are shown as black curves, and parts of the trajectories that may be hard to keep track of are annotated with orange \rightarrow . The three images in the bottom row ((d), (e), and (f)) show examples of unsuccessful movements (using fitness criteria S^2T without recurrent connections), where the limbs failed to reach the target.	55
4.8	Average success rates of controllers based on different fitness criteria. Four sets of experiments (evolution of the neural circuit controller followed by testing) were ran for each fitness criterion. In general, fitness criteria that included T (tool pick-up frequency) did significantly better than those that did not include T (t-test, $n = 4$, $p < 0.01$). Results with S^2T were similar (data not reported here). Note that the target was placed within the limb's reach only $\sim 50\%$ of the time during the trials. See text for details.	57
4.9	Fitness over generations and network size, with or without recurrent connections. S^2T with (blue line) and without recurrent connections (red line) are compared. Top-left: max fitness scores through the generations. Top-right: average fitness scores. Bottom-left: number of total neurons. Bottom-right: number of total degrees (number of connections) in the neural circuits through the generations.	58
4.10	Correlations between the number of recurrent connections (self loop + 2 hops + 3 hops) and success rates. The data points (blue "x"), the linear regression lines, and correlation coefficients (r) are plotted, for the six fitness criteria. For each fitness criteria, the 120 best neural circuits for each generation (generation 1 to 120) were analyzed to see the correlation between the number of recurrent connections and success rates (average of 1,000 trials each). The total number of cycles (number of loops) is found to be positively correlated with success rate. Note that the correlation is even higher for those that included T (tool pick-up frequency) in the fitness.	60

4.11	Internal dynamics of fifteen hidden neurons, their correlations, and matching limb movements. (a) Evolved neural circuit with fitness = DST. (b) Time series correlation matrix of the fifteen hidden neurons. (c) Time-lapse image of the limb behavior. (d) Time series values of the hidden neurons. Event in (c) are marked with dashed lines. See text for details.	62
5.1	REINFORCE algorithm pseudocode [125]	76
5.2	Overview of the actor-critic architecture with the tool-use environment. At time t , reward r_t and state S_t are sampled from the tool-use environment. The observed state S_t are fed into the policy network (the actor with parameter vector θ) and the value network (the critic with parameter vector \mathbf{w}). Next, the policy network produces the stochastic policy $\pi(A_t S_t, \theta)$ that consists of the mean μ and the standard deviation σ vectors of four dimensional normal distributions. The actions A_t for each joint (the motor torque vector for the joints) are sampled from the multidimensional normal distributions. The sampled action vector A_t are fed into the advantage function and used to take the next action. Also, the value network estimates the value of the state $\hat{v}(S_t, \mathbf{w})$ which is fed into the advantage function. The advantage function $A^\pi(S_t, A_t)$ keeps the values of A_t , $\hat{v}(S_t, \mathbf{w})$, and the reward r_t for k time steps, then use them to update the policy and value parameters (θ and \mathbf{w}) using the natural gradient for the trust region update. The environment receives the sampled action vector A_t , take a next step, and produces the new state and reward (S_{t+1} and r_{t+1}). See text in Sec. 5.2.3 for details.	82
5.3	Successful example of the tool-use task, where the screen shots of the subtasks completion are shown. (a) shows initial locations of the tool, object and arm. For each episode, the object location is randomly selected out of the arm reach, the tool location is fixed, and the arm location is either right or left side of the tool. (b) Open the gripper and bend the arm. (c) Reach the tool handle. (d) Grasp the tool. After grasping the tool, the agent need to keep the gripper closed to manipulate the tool. (e) Move the tool (tool end-effector) to the object. (f) Move the object to the bottom bar (target) using the tool. See text for details.	85
5.4	Examples of unsuccessful episodes. In both screen shots (a) and (b), the arm hit the tool and as a result the tool moves away to a distance that the arm cannot reach.	86

- 5.5 Success rate comparison for the best models. The best models from the five different random seeds for each of the four reward schemes were selected and ran for 1,000 episodes with the trained policy networks. An episode was counted as success if the last subtask (Object Reach Target) is satisfied. The bar-plots show the success rates of each reward scheme’s best model. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (success rate: 93%) and *stepwise-no-intermediate* (success rate: 96%) show success rates over 90% where the latter one is slightly higher than the first one. The *one-sparse* and *no-stepwise-all* never completed the task. 89
- 5.6 Mean episode reward for the best models. The best models for each reward schemes were selected and run for 1,000 episodes with the trained policies. The bar-plots show the mean episode reward of each scheme’s best model. For this comparison, the reward scheme *no-stepwise-all* was used to make sure all kinds of rewards are always given even though subtasks are not completed. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (mean reward: 1615.4) and *stepwise-no-intermediate* (mean reward: 1607.3) show much higher reward means than the *one-sparse* (mean reward: 794.2) and *no-stepwise-all* (mean reward: 776.3). The *no-stepwise-all* has a larger standard deviation than *one-sparse* even though they both could not learn the first subtask (Reach Tool) and have similar mean reward. This is because the agent *one-sparse* learning was converged to minimize joint movement (staying without much moving) while the *no-stepwise-all* learning could not converge to any behavior (kind of random moves). 90

- 5.7 Mean episode length for the best models. The best models for each reward scheme were selected and run for 1,000 episodes with the trained policies. An episode length ends early when the episode satisfies the last sub-task (Object Reach Target) before it reaches the maximum time steps (500 steps). The *stepwise-intermediate* (mean length: 135.5 steps) and *stepwise-no-intermediate* (mean length: 136.8 steps) show shorter mean episode length than the maximum steps where the first one shows slightly shorter mean episode length. This may be the reason why the *stepwise-intermediate* scheme shows better mean reward than *stepwise-no-intermediate* in Fig. 5.6 even though the one has lower success rate than the latter one in Fig. 5.5. The best model from *stepwise-intermediate* learned a behavior where the tool end-effector is moved directly towards the object while the *stepwise-no-intermediate* learned a behavior where the tool is moved to the left and right boundary with a full stretch of the arm. However, the first one (*stepwise-intermediate*) makes more mistakes than the latter one (*stepwise-no-intermediate*) when grasping the tool and moving the tool end-effector to the object. In sum, the *stepwise-intermediate* scheme learned to complete the task faster than the *stepwise-no-intermediate*, while the latter learned to perform the task more stably than the former. Note that this behavioral difference did not come from the different reward schemes. The different random seeds played a role in the behavioral differences. We can say that both schemes led to good enough local optima. The *one-sparse* (length mean: 500.0 steps) and *no-stepwise-all* (length mean: 500.0 steps) never completed the tasks. 91
- 5.8 Comparison of all five models (including all different random seeds) for each reward scheme. All five models for each scheme trained with different random seeds were compared by running for 1,000 episodes with the learned policies. The trends are similar to the comparison of the best models, where the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show higher success rate, higher mean episode reward, and shorter mean episode length than the *one-sparse* and the *no-stepwise-all* schemes. The differences between the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes are very subtle. The *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show degraded values (lower success rates, lower mean episode reward, and longer mean episode length) compared to the best models because the models here include unsuccessful learnings. The *one-sparse* and the *no-stepwise-all* schemes have almost the same values (success rates, mean episode reward, and mean episode length) to the best models. 92

- 5.9 Learning curve for different random seeds under *stepwise composite reward with intermediate elements (stepwise-intermediate)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 24,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training were 60M at most. The reason why different random seeds lead to different total iterations is that some random seeds (the blue and red learning curves) successfully learned the tasks and complete a task early as in the Fig. 5.9b, while some others (the orange and sky-blue learning curves) could not learn the task successfully so the episode lengths were equal to the maximum episode steps (500 steps). (a) Mean episode reward during training. The two models (the blue and red learning curves) learned the task successfully around 12,000 iterations and reached about 1400 reward. The pink case reached an intermediate performance and stayed at the plateau. The orange and sky-blue cases could not learn the task successfully. (b) Mean episode length during training. Successful cases (the blue and red learning curves) show shorter episode lengths as they show higher rewards, while the others show longer episode length (the pink learning curve) or maximum episode length (500 steps) as they could not reach the target for early episode termination. 95
- 5.10 Learning curve for different random seeds under *stepwise composite reward without intermediate elements (stepwise-no-intermediate)*: Mean episode reward and mean episode length during training with five different random seeds. Different colors mean different random seeds. The model instances were trained for 24,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training are 60M at most. The pink case successfully learned the task at around 7,000 iterations. The orange case learned gradually and reached the successful learning state at around 23,000 iterations. The others could not learn the task. The blue and sky-blue cases could not get out of the plateau, where they could learn the first two subtasks (i.e. Reach-Tool and Grasp-Tool), but could not learn to the other two subtasks (i.e. Tool-Reach-Object and Object-Reach-Target). The red case could not show much learning, where it could learn to make the gripper close to the tool handle, but closes the gripper too early so that it was not able to grasp the tool. a) Mean episode reward during training. (b) Mean episode length during training. The graph shows inverse correlation with the graph in (a). See text for details. 98

- 5.11 Learning curve for different random seeds under *one sparse reward (one-sparse)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 23,000 iterations at most (2 days and 12 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training are 57.5M at most. None of the models could learn even to make the gripper close to the tool. It seems that one of the episode was lucky to complete the task (the high spike in the red learning curve) at around 900 iterations, but the one lucky trial could not facilitate further progress. (a) Mean episode reward during training. Obviously, none of the models could get reward (except the red case that got lucky). The agents learned to reduce the torque penalty, but could not learn any of the subtasks. (b) Mean episode length during training. Obviously, none of them learned to complete the task so that all the episodes lasted to the maximum step of 500. See text for details. 99
- 5.12 Learning curve for different random seeds under *all reward elements without stepwise order (no-stepwise-all)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 22,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training were 55M at most. Since all the rewards elements are always provided in this scheme, the models always show some amounts of reward, but they are all fluctuating between around 700 and 850, and none of them showed progress. Interestingly, none of them could not even learn to move the gripper close to the tool even though the continuous reward element for the distance between the gripper and the tool is provided. This indicate that presenting extra rewards such as the distance between the tool and the object and/or the one between the object and the target before learning the first subtask (reach) affects the learning negatively. (a) Mean episode reward during the training. (b) Mean episode length during the training. None of them learned the first subtask (Reach-Tool) and completed the task before reaching the maximum time step of 500. See text for details. 101

1. INTRODUCTION

Learning through the sensorimotor loop is essential for intelligent agents. Intelligent agents, whether they are biological or artificial, are always driven by sensory inputs such as proprioceptive, visual, auditory, tactile, olfactory, vestibular, and gustatory input. Sensory inputs directly and indirectly drive sequence of motor actions that lead to changes in the agent's body and the status of the world, which is followed by updated sensory inputs. The important role of sensorimotor learning has been discussed in many studies [123, 13, 15, 95, 35, 56, 152, 110, 92]. However, several important aspects of sensorimotor learning in the brain are still hardly understood, such as the development of behavior maps, the influence of internal dynamics on external behaviors, and the emergence of tool-use capabilities.

In this dissertation, I will address three questions trying to probe the nature of sensorimotor learning: (1) How can a sensorimotor agent understand its own body by developing a cortical map of its motor actions?; (2) How can predictive brain internal dynamics play an important role in external behavior (authorship of actions)?; (3) How tool-use emerges in a sensorimotor system interacting with the environments? The backgrounds, goals, approaches, and contributions for each topic are briefly described in the rest of this section.

1.1 Motivation

(1) A sensorimotor agent needs to understand its own body by developing a map of its own behavior in the brain. In an experimental study, it was found that the motor cortex in the macaque brain forms a topographical map of complex behaviors, where the final posture of the movements form a topologically organized map (Graziano et al.[47]). For example, one part of the motor cortex maps the target location in reaching behavior, where the map shows a topological organization (nearby neurons represent nearby target locations in the environment). Motivated by this finding from the biological experiments, in this dissertation,

I investigate how a motormap in the cortex of the brain can be self-organized. Specifically, I will investigate the possibility that a motor map in the cortex can be developed based on the same cortical learning process as the visual and tactile maps in the cortex (Chapter 2).

(2) The previous computer simulation studies showed that neural network controllers with more predictable internal state dynamics can attain higher performance in harsher or changing environments, which also indicates higher chance of survival in evolution [68, 29, 25]. The implication of this finding is profound since it implies that the internal state's properties can affect the external behavioral performance especially in changing environment, and predictable internal state dynamics could be a necessarily condition for intelligent agents in the evolutionary pathway to have authorship of actions and to adapt themselves to changing environment. However, there is a missing link to connect between the findings from the simulation studies and the real brain. The findings from the simulation study do not necessarily mean that the internal state dynamics affects the external behavior (authorship of actions) in the biological brain as well. To fill the gap, I investigated the role of predictability in internal state dynamics in the brain by analyzing the human EEG data (Chapter 3).

(3) Tool-use [52], which requires high levels of sensorimotor skill learning and problem solving capabilities, is one of the salient indicators of intelligence along with communication (language) [117] and logic [4]. Communication and logical inference have been extensively investigated in artificial intelligence (AI). However, tool-use is still largely under-developed in the field of AI [27]. Through our literature search (details are in Chapter 4), we found that there are two gaps in tool-use in AI and robotics. First, most works depend on some degree of designer knowledge regarding tool-use and motor control. Furthermore, tool-use tasks that require multiple subtasks to be completed in specific order are almost non-existent in deep reinforcement learning (RL) literature for developments and benchmarks, even though deep RL has demonstrated good performance in some control tasks in recent years.

In this project, I present environments and sensorimotor systems where the agents can

adapt to use simple or complex tools based on minimal task knowledge. Specifically, I present a two step approach. I will first evolve neural network controllers for simple tool-use behavior in reaching tasks with minimal task knowledge, followed by analysis of the evolved networks (Chapter 4). Then, as a second step, I will implement a more complex tool-use task such as dragging an object using a tool in a physics simulation environment, and demonstrate a deep RL method and reward shaping methodology to learn the complex tool-use task successfully (Chapter 5).

1.2 Approach

To achieve the above objectives, (1) a computational model of cortical development called GCAL (Gain Control, Adaptation, Laterally connected), which is basically an unsupervised learning algorithm, mimicking synaptic plasticity in the cortex of brain, was used to develop a topographical map of complex motor actions (Chapter 2); (2) the predictability of Inter-peak interval (IPI) was analyzed from the internal brain dynamics data (human EEG data) and its relation to agent's own control of external behaviors (authorship of actions) is discussed (Chapter 3); (3) a combination of neural networks, genetic optimization, and deep reinforcement learning will be used for designing environments and sensorimotor agents where they can adapt to use simple or complex tools based on minimal task knowledge. A physics simulation platform (MuJoCo) will be used for physics simulations (Chapter 4 and 5).

1.3 Organization of this Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, I discuss the investigation about how a motormap in the cortex of the brain can be self-organized. Specifically, I will present the possibility that a motor map in the cortex can be developed based on the same cortical learning process as the visual and tactile maps in the cortex.

In Chapter 3, the implication of previous computer simulation studies regarding the

predictable internal state dynamics and the missing link between the simulation study and the brain will be discussed. Then, the investigation about the role of predictability of internal state dynamics in the brain by analyzing the human EEG data will be presented.

In Chapter 4 and 5, the gaps regarding tool-use in AI and robotics and also in deep reinforcement learning will be discussed. Then, the task environments and sensorimotor systems will be presented where the agents can adapt to use simple or complex tools. Specifically, as a first step, in Chapter 4, I will present evolving neural network controllers for simple tool-use behavior in reaching tasks with minimal task knowledge, followed by analysis of the evolved networks. Then, in Chapter 5, as a second step, I will present a more complex tool-use task such as dragging an object using a tool in a physics simulation environment, and demonstrate a deep RL method and reward shaping methodology to learn the complex tool-use task successfully.

Finally, in Chapter 6, I will summarize the contributions and conclude the dissertation.

2. DEVELOPMENT: SELF-ORGANIZATION OF THE MOTOR MAP IN THE CORTEX

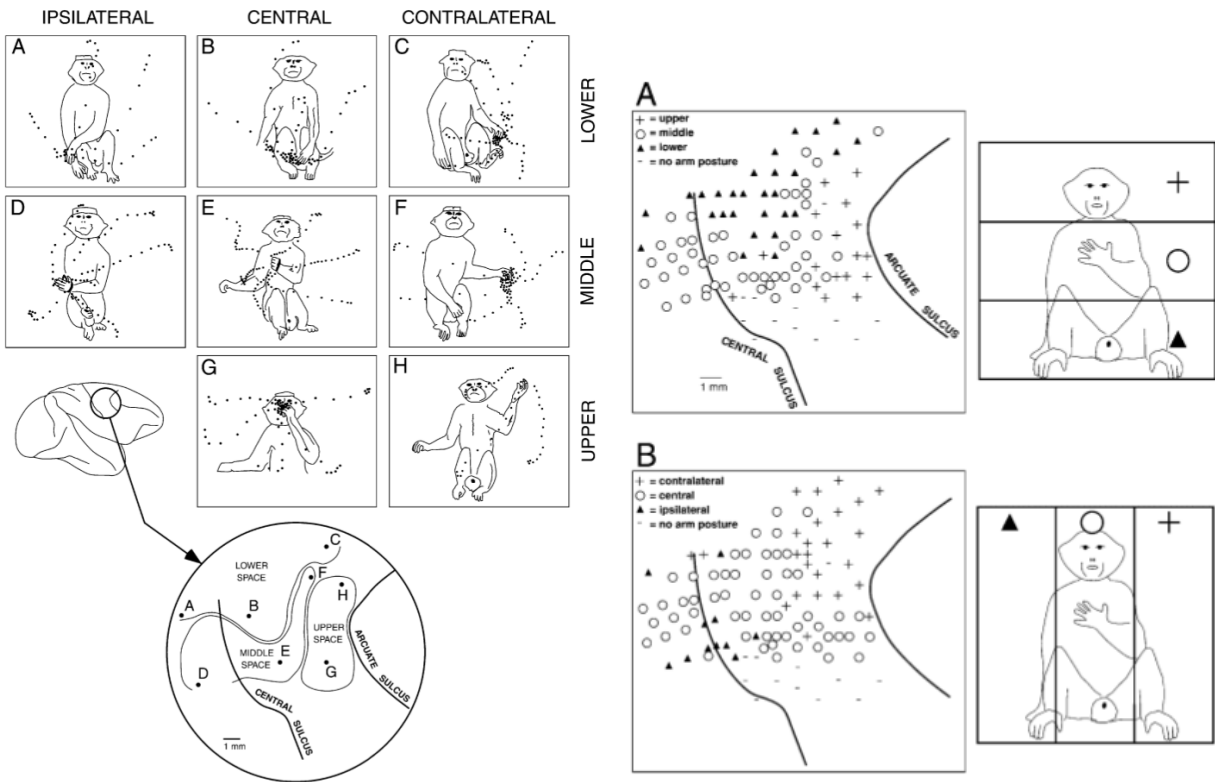
2.1 Background

In recent studies, Graziano et al. found that the motor cortex in the macaque brain forms a topographical map of complex behaviors [46], where the final target posture of the movements form an organized map. As we can see in the Fig. 2.1a, the monkey's target location in the reaching behavior evoked by extended electrical microstimulation on a certain location of the motor cortex was always the same, regardless of the initial hand position. Furthermore, the target location forms an organized map on the motor cortex, where ventral and anterior areas corresponded to the target locations in the upper space of the body, whereas dorsal and posterior areas in the motor cortex corresponded to target locations in the lower space of the body as shown in Fig. 2.1b. Based on these findings, our question is how such motor maps are formed in the cortex through the development period.

In existing works, simulation studies were conducted to mimic the development of visual and tactile maps in the cortex. The visual cortical neuron's receptive fields (RFs) and their map were developed computationally using a self-organizing map model of the cortex (the LISSOM model) by Miikkulainen et al. [81]. Park et al. showed that both visual RFs and tactile RFs can be derived by training the same self-organizing map model of the cortex with different types of inputs (natural-scene images and texture images, respectively) [91].

In this paper, we investigate the possibility that a motor map in the cortex can be

Part of this section is reprinted with permission from Jaewook Yoo, Jinho Choi, and Yoonsuck Choe, "Development of target reaching gesture map in the cortex and its relation to the motor map: A simulation study." In *Advances in Self-Organizing Maps and Learning Vector Quantization: Proceedings of the 10th International Workshop, WSOM 2014, Mittweida, Germany, July 2-4, 2014*, pages 187-197. © 2014 Springer, Heidelberg.



(a) Eight example posture illustrating the topographic(b) Topography of hand and arm postures in the pre-
 map found in the precentral cortex of monkey. central gyrus based on 201 stimulation sites in monkey.

Figure 2.1: The topographic map found in precentral cortex of Monkey. (a) The enlarged view at the bottom shows the sites of the electrical microstimulation. The movements shown in the rectangles A–H were evoked by stimulating the sites A–H in the enlarged circle at the bottom. The stimulation of the right side of the brain caused mainly the left side of the body (left arm to move). (b) A shows the distribution of hand positions along the vertical axis, which are upper, middle, and lower space. After each stimulation, the evoked final target positions were used to categorize the site. B shows the distribution of hand positions along the horizontal axis, which are contralateral (right when using left hand), central, and ipsilateral (left when using left hand) space. Adapted from [46].

developed based on the same cortical learning process as the visual and tactile maps in the cortex. A self-organizing map model of the cortex (the GCAL model[12][70]: a simplified yet enhanced version of the LISSOM model[81]) was trained with two-joint arm movements (2 DOF) on a 2D plane, which were subsequently encoded as a time-lapse image (cf. Motion History Images [18]) where time was encoded as the pixel intensity. We investigated if the experiment can give rise to a motor map organization similar to Fig. 2.1.

This chapter is organized as follows. The related works are reviewed in Section 2.2. Next, the GCAL model will be explained in Section 2.3. Section 2.4 will explain the simulation platform and the procedure for the experiments. The results are presented in Section 2.5. Discussion and conclusion are presented in Section 2.6 and 2.7, respectively.

2.2 Related Work

There is comparatively little work on how motor map is formed in the cortex. Recently, several related studies were conducted, where a simulation study for the motor map clustering of monkey using a standard self-organized map (SOM) learning with encoded movements, and a multi-modal reinforcement learning algorithm to form a map according to behavioral similarity.

Aflalo and Graziano [2] showed a computational simulation where topographic map organization emerged based on three constraints. The three constraints were: (1) the body parts that were being moved for movements, (2) the position reached in Cartesian space, and (3) the ethological (behavioral) category to which the movement belonged. Encoded movements (body parts, hand coordinates, and behavioral category) were used as inputs for training. The initial somatotopic body map from the literature was used to initialize the model. A standard Self-Organized Map (SOM) learning [65] was used for the motor map clustering. However, their map configuration through the SOM learning is based on a purely computational abstraction since their experiment with the learning algorithm did not consider the

neural connectivity or plasticity in the cortex. Also, the initial somatotopic body map which was set up to already represent a rough motor map of the adult brain significantly affected the final configuration of the map.

Ring et al. introduced a new approach to address the problem of continual learning [103][102], which was inspired by recent research on the motor cortex [46]. Their system modules, called *mot*, are self-organized in a two-dimensional map according to behavioral similarity. However, their method was based on a multi-modal reinforcement learning algorithm, and did not consider the neural underpinnings. Their aim in the study was to improve learning performance through their new approach, not to understand how the motor map in the cortex is developed.

While several related studies have been conducted, there is a lack of studies for fine-grained, biologically plausible motor map development in the cortex.

2.3 The GCAL Model

We trained the GCAL (Gain Control, Adaptation, Laterally connected) model of cortical development to investigate motor map development in the cortex with 2 DOF arm movements in 2D plane. The GCAL model is a simplified, but more robust and more realistic version of the LISSOM model, which has been developed recently by Bednar et al. [12][70]. The GCAL model was designed to remove some of the artificial limitations and biologically unrealistic features of the LISSOM model.

GCAL is a self-organizing map model of the visual cortex [12][70]. Even though GCAL was originally developed to model the visual cortex, it is actually a more general model of how the cortex organizes to represent correlations in the sensory input. Therefore, sensory modalities other than vision should work with GCAL. For example, earlier work with LISSOM, a precursor of GCAL, was used to model somatosensory cortex development [91, 142].

In our GCAL experiments, we decreased the retina size to 2.0 to fit the input image size

(80×80 pixels) and enlarged the projection area (radius: 1.5) to project all parts of the arm movements. The sizes of LGN ON and LGN OFF maps in the thalamus (lateral geniculate nucleus) and their projection size were the same as that of the retina. The radius of the projection area for LGN ON (and LGN OFF) was calculated as $r = \frac{v+l}{2}$, where r , v , and l indicate the radius of the projection area for LGN ON (or LGN OFF), the V1 area, and the LGN ON (or LGN OFF) sheet size, respectively. This way, the arm movements can be projected to the V1 level without cropping. Also, the other parameters were adjusted according to the sheets and the projection sizes. *Note that in the following we will use the GCAL terminology of retina, LGN, and V1 to refer to the sensory surface, thalamus, and cortex, respectively.*

The following description of the GCAL model closely follows [12][70]. The basic GCAL model is composed of four two-dimensional sheets (three levels) of neural units, including the retinal photoreceptor (input) and the ON and OFF channel of RGC/LGN (retinal ganglion cells and the lateral geniculate nucleus), the pathway from the photoreceptors to V1 area. Fig. 2.2 shows the architecture of GCAL we used.

The GCAL training consists of four steps overall as below.

1. At each iteration (input), the retina (sheet) is activated by the time-lapse image of the 2-DOF arm movement.
2. LGN ON and LGN OFF sheets are activated according to the connection weights between the retina and the LGN ON and LGN OFF sheets. Also, the lateral connections from other neurons in the LGN ON and LGN OFF sheets affect the activations. The activation level η for a unit at position j in an RGC/LGN sheet L at time $t + \delta t$ is defined as:

$$\eta_{j,L}(t + \delta t) = f \left(\gamma L \sum_{i \in F_j} \psi_{i,P}(t) \omega_{i,j} \right) \quad (2.1)$$

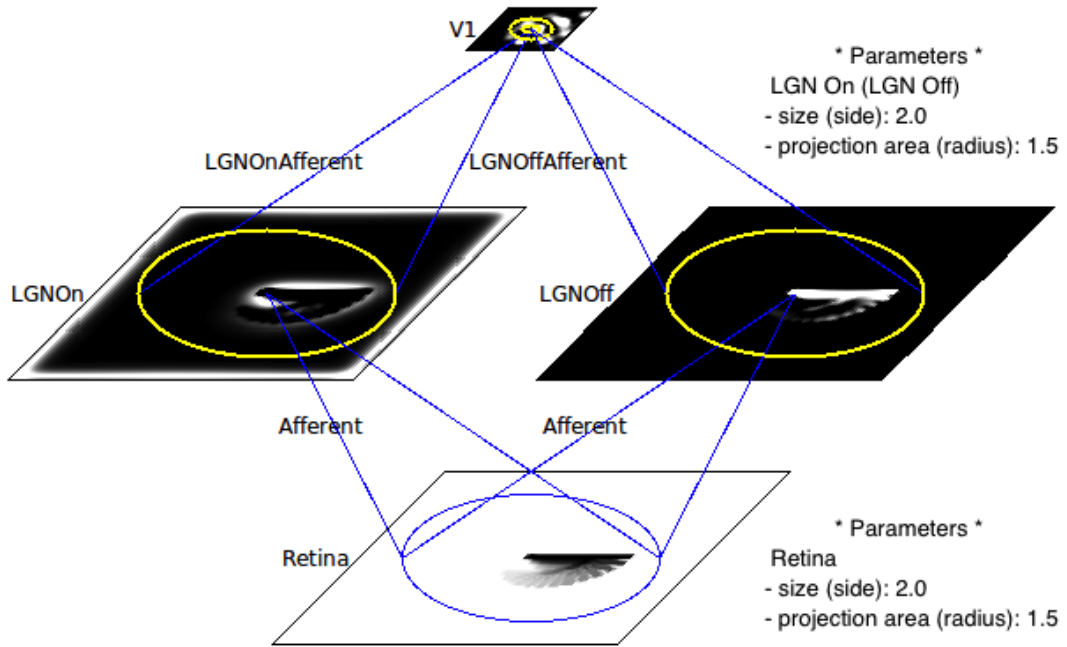


Figure 2.2: The GCAL architecture. In the model, the retina size was increased to 2.0 (side) to fit the input image size (80×80 pixels) and the projection area enlarged to 1.5 (radius) to project all parts of the arm movements. *Note that in the text we will use the GCAL terminology of retina, LGN, and V1 to refer to the sensory surface, thalamus, and cortex in our gesture map model, respectively.*

where the activation function f is a half-wave rectifying function. The terms γL , $\psi_{i,P}$, and ω_{ij} are defined as follows:

- γL is an arbitrary multiplier for the overall strength of connections from the retina sheet to the LGN sheet.
 - $\psi_{i,P}$ is the activation of unit i in the two-dimensional array of neurons on the retina sheet from which LGN unit j receives input (its connection field F_j).
 - ω_{ij} is the connection weight from photoreceptor weight from the retina i to LGN unit j .
3. V1 sheet is activated by three different types of connections: 1) the afferent connection from the LGN ON and LGN OFF sheets ($p = A$), 2) the recurrent lateral excitatory connection ($p = E$), and 3) the recurrent lateral inhibitory connection from other neurons in V1 sheet ($p = I$). The V1 activation is settled through the lateral interactions. The contribution, X_{jp} , to the activation of unit j from each lateral projection type ($p = E, I$) is then updated for the settling steps as:

$$X_{jp}(t + \delta t) = \sum_{i \in F_{jp}} \eta_{i,V}(t) \omega_{ij,p} \quad (2.2)$$

where $\eta_{i,V}$ indicates the activation of unit i taken from the set of neurons in V1 that connect to unit j . F_j is its connection field. The weight $\omega_{ij,p}$ is for the connections from unit i in V1 to unit j in V1 for the projection p . The afferent activity ($p = A$) remains constant during this setting of the lateral activity.

4. V1 neuron's activation level is calculated over time by a running average (smoothed exponential average), and the threshold automatically adjusted through a homeostatic mechanism.

5. LGN to V1 and V1 lateral connections are adjusted using a normalized Hebbian learning rule.

$$\omega_{ij,p}(t) = \frac{\omega_{ij,p}(t-1) + \alpha\eta_j\eta_i}{\sum_k (\omega_{kj,p}(t-1) + \alpha\eta_j\eta_k)} \quad (2.3)$$

where for unit j , α is the Hebbian learning rate for the afferent connection field F_j .

2.4 Experiment

We trained the GCAL model in Fig.2.2 with target reaching behavior of the two-joint arm on a 2D plane. We generated 20,000 movements in which each started from a random location (posture) and moved towards one of the 24 predefined target locations (postures). These input movements simulate the monkey’s arm movement in Fig 2.1a. Our main question was if the model can learn a target reaching gesture map such that the map has the characteristics of the motor map in Fig. 2.1a and Fig. 2.1b. Details about the experiment platform, generating movements, and experiment procedures are as follows.

2.4.1 Experiment Platform

We ran the experiments on a Desktop PC (CPU: Intel Core 2 Duo 3.16GHz, Memory: 16GB) and Laptop (CPU: Intel Core i7 2 GHz, Memory: 8GB). Both machines ran on Ubuntu 10.04 (32bit). The installed Topographica version was 0.9.7. The python version was 2.6.5, and the gcc/g++ version 4.4.3.

For training the GCAL model, we mainly used the Topographica neural map simulator package, developed by Bednar et al. [81]. Topographica is a simulator for topographic maps in any two-dimensional cortical or subcortical region, such as visual, auditory, somatosensory, proprioceptive, and motor maps plus the relevant parts of the external environment [81]. The simulator is mainly written in Python, which makes it easily extendable and customizable according to the users’ needs. The simulator is freely available including the full source code

at <http://topographica.org>.

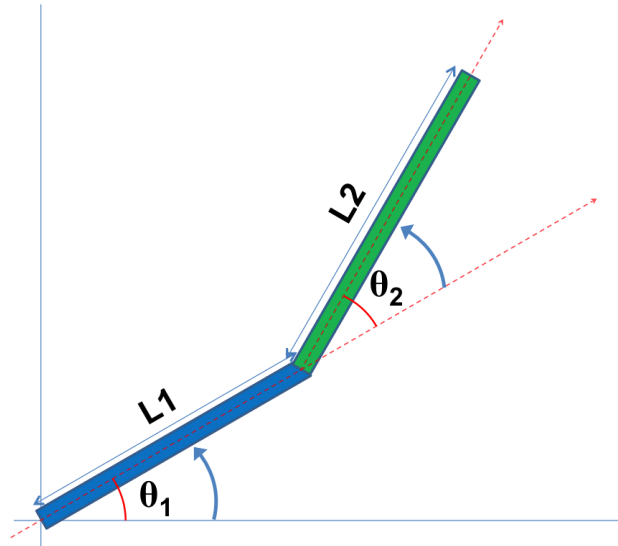
2.4.2 *Generating movements*

Two-joint arm movements were generated on a 2D plane and used as inputs for the GCAL model training. Each of the 20,000 different arm movements started at a random location (posture) and moved towards one of the 24 predefined target locations (postures). These generated movements represent the arm movements of the monkey described in Fig. 2.1a and Fig. 2.1b.

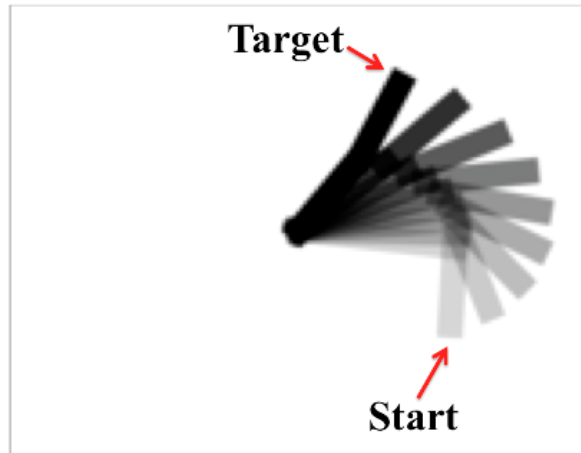
The arm consisted of two joints J1 (θ_1) and J2 (θ_2) in which the length ratio of the arm L1 : L2 is 1.6 : 1 (Fig. 2.3a). For each movement, first randomly pick initial angles for θ_1 and θ_2 (between $-180^\circ \sim 180^\circ$) and the 24 target locations as shown in Fig. 2.4. Then, J1 (θ_1) and J2 (θ_2) are changed toward the target locations from the initial angles either by 5 or 10 degrees each step, until they reach to the target posture. After each step of the angle update, the posture of the arm was plotted on the same sheet but with different opacity. The intensity was increased over time by 20% (Fig. 2.3b). Fig. 2.4 shows the examples of the generated arm movements. The 24 predefined target locations consisted of 16 distal locations (Fig. 2.4(a)) and 8 proximal locations (Fig. 2.4(b)). Note that in generating these motion patterns, we did not consider the natural movement statistics of the monkey's arm, largely due to the lack of such data.

2.4.3 *Experiment Procedure*

After generating 20,000 random reaching movements encoded as a time-lapse image, we trained the GCAL model with these inputs. In each iteration, one of the 20,000 inputs was randomly picked for training. The density of the two LGN sheets and the single V1 sheet were 24×24 and 48×48 , respectively. The parameters of the model for training were based on the default parameters in the Topographica package except some parameters such as retina sizes as described in Section 2.3. Once the training is done, we analyzed the resulting



(a) Kinematics of the arm



(b) Time-lapse encoded arm movement

Figure 2.3: The kinematics and movement of the two-joint arm. (a) The arm consists of two joint $J1 (\theta_1) : J2 (\theta_2)$, and the arm $L1 : L2$ (with the length ratio $1.6 : 1$). The θ_1 and θ_2 are randomly picked initially and change toward the target. (b) The arm movement is encoded as time-lapse image where time is encoded as the pixel intensity. The darkest one is the target (most recent) posture.

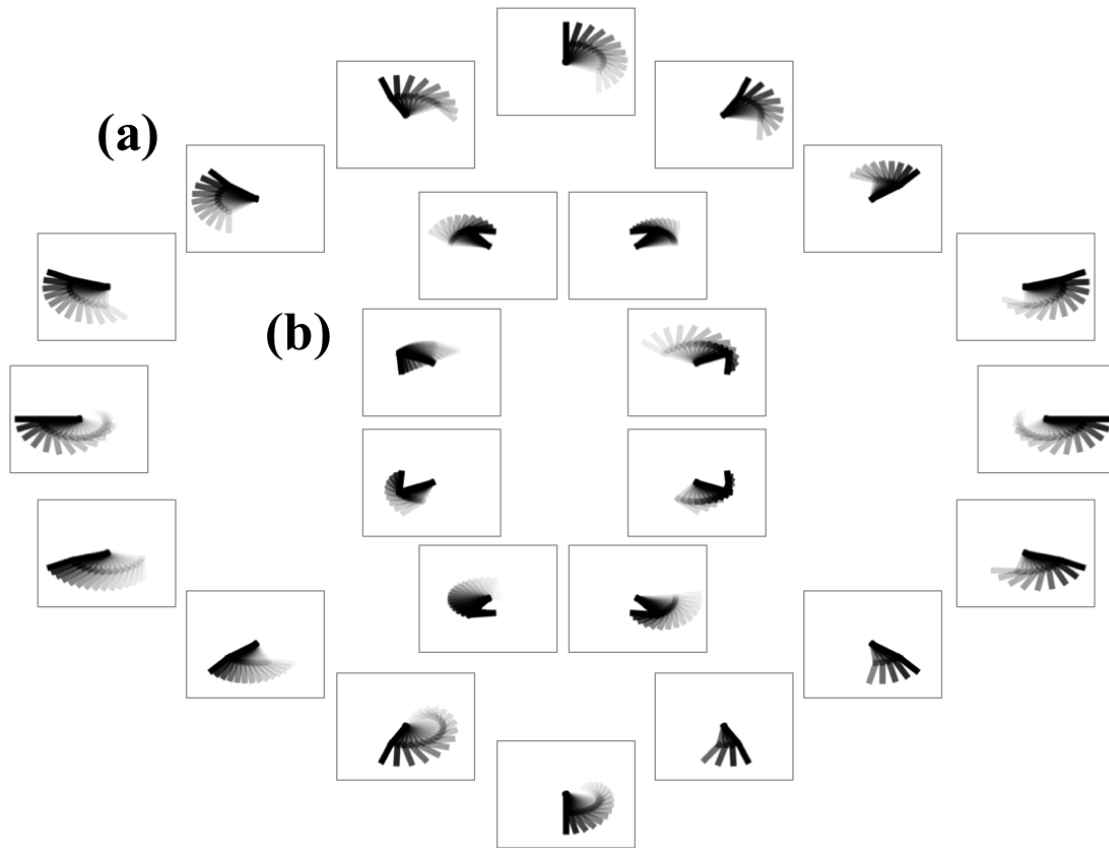


Figure 2.4: Examples of movements with 24 target locations. Starting from a random posture, move toward to one of 24 target locations (postures). The movement over time is expressed using different pixel intensity (darker = more recent). (a) Example movements with 16 distal target locations. (b) Example movements with 8 proximal target locations. These movements simulate the arm movements in the experimental literature (Fig.2.1a). Note: For the same target location, many different time-lapse images were generated by varying the initial posture.

map, with a focus on the LGN to V1 afferent connection patterns.

2.5 Results

2.5.1 Local topography based on target location similarity

The resulting gesture map is shown in Fig. 2.5. As we can see, the map is topologically ordered according to the target locations, where nearby locations (neurons) of the map represent nearby target locations (end-effector locations of final postures). For example, we can see that the similar target locations are clustered in the areas of top-left, top-right, bottom-left, bottom-right, center-left, center-right, and so on. In Fig. 2.6(a), each arrow of the grids shows the orientation and the distance of the target locations from the center. The vectors (arrows) with similar lengths and orientations represent similar target distance and angle.

2.5.2 Global topographic order

The resulting gesture map show global topographic order. The color maps of the horizontal and the vertical components of the vector field in Fig. 2.6(a) are shown in Fig. 2.6(b). As we can see, the vectors (the target locations) show horizontal order (Fig. 2.6(b), top) and vertical order (Fig. 2.6(b), bottom), which is comparable to the findings reported in the experimental literature (Fig. 2.1). Some neighboring vectors show opposite directions in Fig. 2.6(a), but this is consistent with the biological observations. As we can see in Fig. 2.1b, some adjacent stimulation sites in the precentral gyrus in the monkey show targets in the opposite directions such as upper vs. lower, or left vs. right arm postures. Look for + and ▲ located right next to each other in Fig. 2.1b.

2.6 Discussion

The main contribution of this paper is the use of a general cortical development model (GCAL) to show how fine-grained target reaching gesture maps can be learned, based on

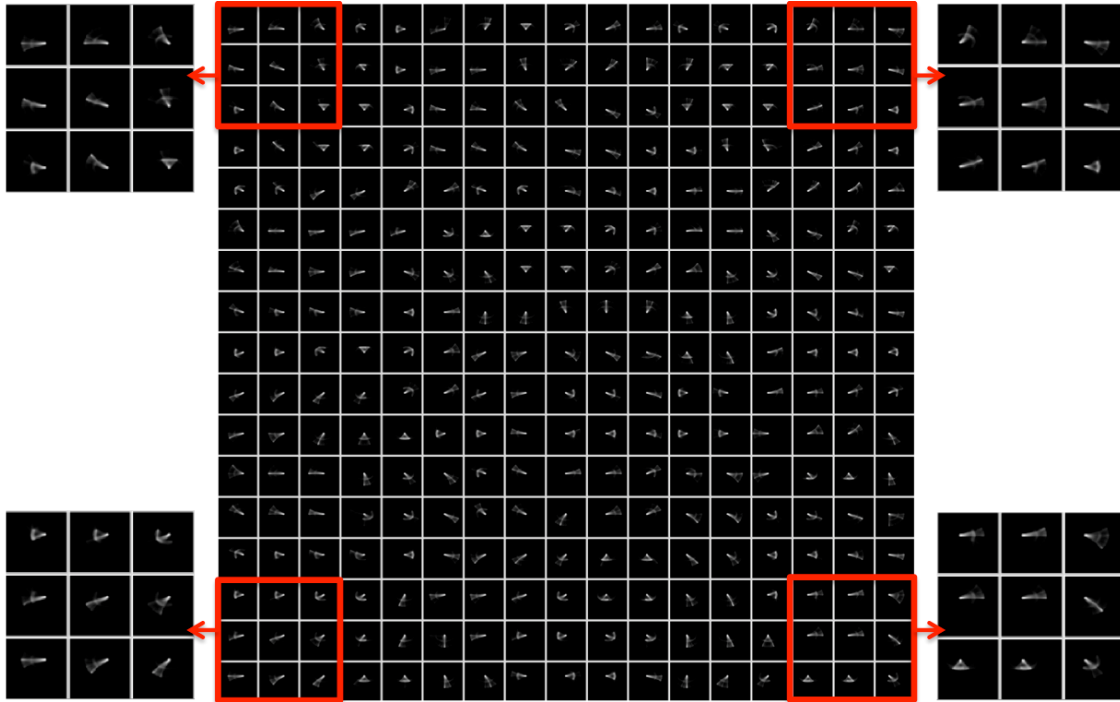
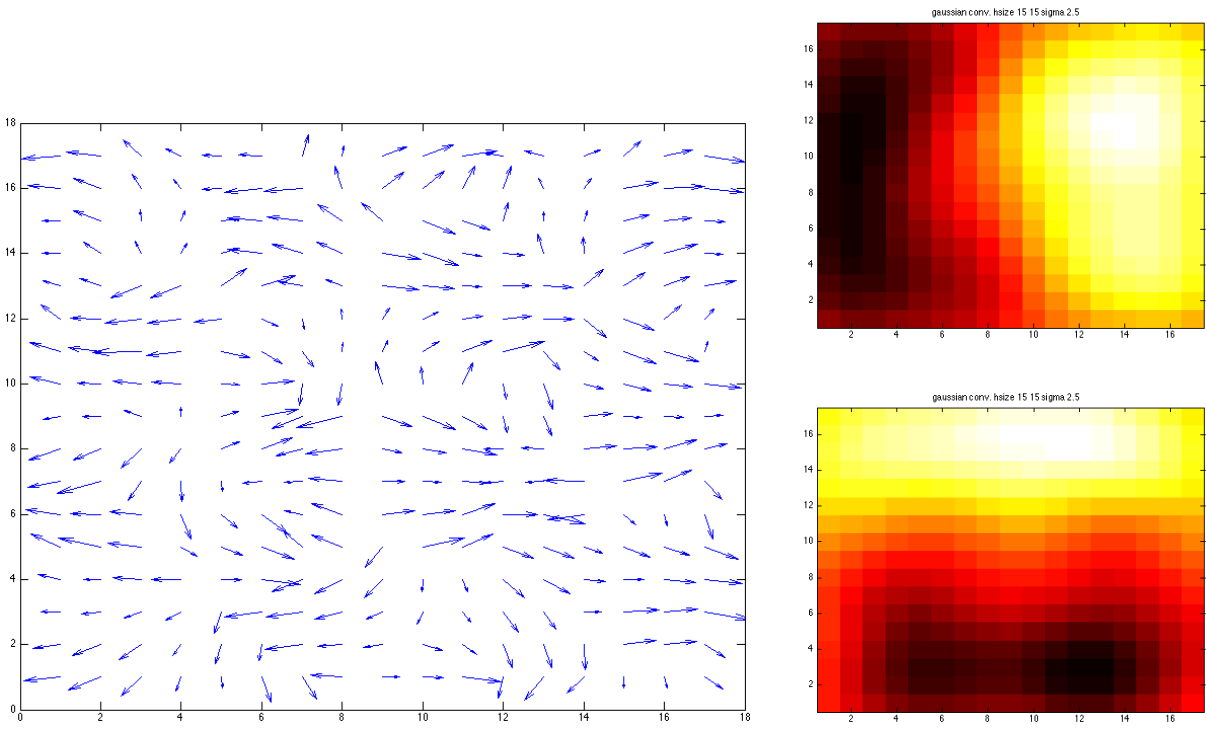


Figure 2.5: The resulting gesture maps of LGN OFF to V1 projection. 17×17 RFs are plotted from 48×48 cortex density to see the details of them. The enlarged views show the zoomed in views of 3×3 RFs at each corner. Note: LGN OFF and LGN ON patterns are exact inverses of each other and thus contain the same information. The learned projections to V1 from these two sheets were similar as a result, so here we only showed the LGN OFF to V1 projections which is easier to visually inspect.



(a) Target vector map

(b) Horizontal (top) and vertical (bottom) components

Figure 2.6: (a) Target vectors estimated from the resulting gesture map receptive fields (Fig. 2.5). The direction and the length of each arrow show the target location's direction and the distance from the center. (b) The color maps of the horizontal and the vertical components of the vectors in (a): bright=high (right, up), dark=low (left, down). The color maps were convolved with a Gaussian filter of size 15×15 pixels and sigma 2.5 to show more clearly the global order.

realistic arm reaching behavior. An immediate limitation is that the input itself was not a dynamic pattern of movement (i.e., it was just a static time-lapse image). However, as shown by Miikkulainen et al. [81], addition of multiple thalamus sheets with varying delay can address this issue. Miikkulainen et al. [81] used such a configuration to learn visual (motion) direction sensitivity in V1.

2.7 Contribution

In this work, we developed a target reaching gesture map using a biologically motivated self-organizing map model of the cortex (GCAL model, a simplified yet enhanced version of the LISSOM model) with two-joint arm movements as input. The resulting gesture map showed a global topographic order based on the target locations. The map is comparable to the motor map reported in the experimental study [46] (Fig. 2.1a and Fig. 2.1b). Although our simulations were based on a sensory cortical map development framework, the results suggest that it could be easily adapted to transition into motor map development. Our work is an important first step toward a fully motor-based motor map development, and we expect the findings reported in this chapter to help us better understand the general nature of cortical map development in a sensorimotor agent, not just for the sensory but also for the motor modality.

3. INTERNAL DYNAMICS: PREDICTABLE INTERNAL BRAIN DYNAMICS AND ITS ROLE IN AUTHORSHIP OF ACTIONS

3.1 Introduction

Predictive function is an important aspect of intelligent robots and agents. Along with prediction, internal states of an intelligent agent have also begun to attract interest among researchers. For example, it was shown that the central nervous system has an internal model in the cerebellum which can be used to predict sensorimotor behaviors [145, 146, 144, 61]. Anticipatory systems were proposed where internal models of the agents' themselves and environments were used to predict the future for controlling their bodies in the present [104]. However, most of the studies regarding prediction and/or internal states have focused on using or considering the internal models (or states) as a method for prediction, but not focused on predictability of internal state itself.

The predictability of internal state dynamics and their effects on external behaviors (authorship of actions) were investigated in the previous computer simulation studies [68, 29, 25]. In those studies, the internal states of a neural network were defined as the activation levels or patterns of the hidden perceptrons [10] since the state of a neural system can be described with the neuronal activation levels or patterns. To investigate internal state dynamics, the researchers evolved recurrent neural network controllers in a dynamic task where prediction is not an immediate fitness criteria. A 2D pole balancing task was used for the experiments where the task goal is to keep the pole straight as long as possible by controlling the cart attached to the base of the pole in a 2D plane (Fig. 3.1a). Among the evolved controllers

Part of this section is reprinted with permission from Jaewook Yoo, Jaerock Kwon, and Yoonsuck Choe, "Predictable internal brain dynamics in EEG and its relation to conscious states," *Frontiers in Neurorobotics*, 8:18, 2014. c 2014 Frontiers.

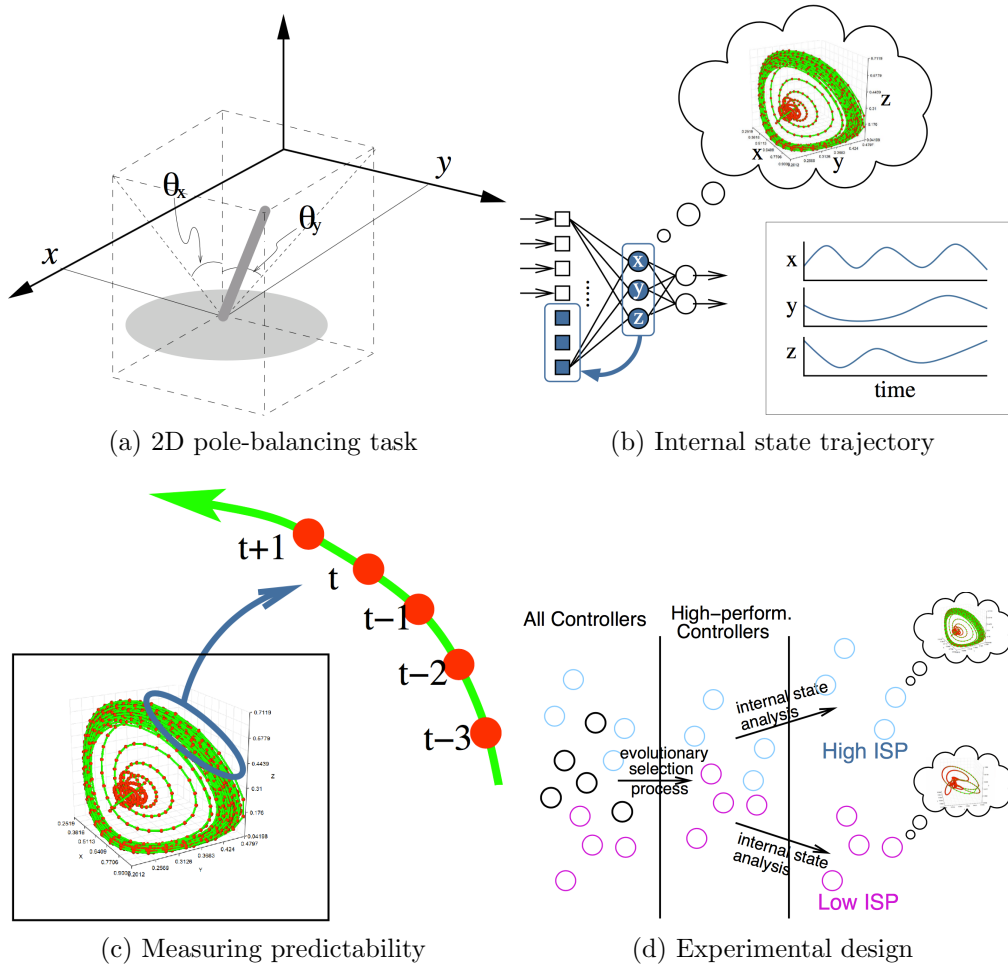


Figure 3.1: Prediction of Internal state trajectory in recurrent neural network controllers for a pole-balancing task. (a) 2D Pole balancing task, where x and y are the coordinates of the cart, and θ_x and θ_y are the angles of the pole from the z -axis. (b) A recurrent neural network controller for the 2D pole balancing task (lower left). The hidden perceptrons' activations (internal state) over time (lower right). A 3D plot of the internal state trajectory (upper right). (c) Measuring predictability of the internal state trajectory, where measuring the prediction error on a next activation value ($t + 1$), given several past activation values as inputs ($t - 3$ to t) (d) The experimental design shows the population of recurrent neural network controllers (left), selecting high-perform controllers (middle), and post-selection analysis by means of Internal State Predictability (ISP) (right). Each controller that passed the selection state has equal task performance, but the analysis of ISP shows that some with high ISP and other with much less ISP. Interestingly, with a harder pole balancing task (by increasing the initial tilt angle of the pole), the controllers with high ISP mostly retain their performance, but those with low ISP lose most of their performance. Adapted from our recent work [69].

that passed the performance threshold (Fig. 3.1d, middle), the predictability of the internal state dynamics (i.e. predictability in the hidden perceptrons activation dynamics over time) were measured to see if high internal state predictability could emerge spontaneously (Fig. 3.1b and 3.1c), just based on the task requirement that does not include the internal state predictability. To measure the predictability of the internal state dynamics, supervised learning was used, where given past n activation values of the hidden units (i.e $n=4$ in the Fig. 3.1c) as inputs, they measured the errors of the predicted values of the next time step. By sliding this window along all the internal state trajectories, the errors of the predicted values were calculated. The lower errors in the prediction means higher predictability. As a result of measuring the internal state predictability from those controllers with high performance for the pole balancing task (Fig. 3.1d, middle), some have highly predictable internal state dynamics and others have much less one. Again, all the controllers, whether with high or low Internal State Predictability (ISP), passed the same performance threshold for the task given the same task from the training. However, it is interesting to find that if the authors made the task harder by increasing the initial tilt angle of the pole, the controllers with high ISP mostly retained their performance, but those with low ISP could not keep their performance. They also analyzed the behavior trajectories of x , y positions and pole angles from the controllers, and found that higher ISP does not necessarily mean simple behaviors. Namely, even controllers with high ISP may have complex behaviors, and those with low ISP may have simple behaviors as well.

To sum up, the previous computer simulation studies above showed that neural network controllers with higher predictable internal state dynamics can attain higher performance in harsher or changing environments, which also indicates higher chance of survival in evolution. Also, the studies showed that the increased performance of the high ISP controllers does not necessarily due to simpler (or predictable) behaviors. Such predictable internal state dynamics could lead to prediction of external behavior and eventually to authorship of actions

because a distinct trait of one’s own actions is always perfectly predictable. ‘Authorship of actions’ means that an agent can distinguish its own motor actions from those of others [51, 16, 11]. For example, humans know when we are the cause of our own voluntary motor actions and take responsibility for the effects. Authorship of actions has been explained by a feed-forward processing (we predict that we are to do an action and then observe the consequences) and by a requirement for tight temporal binding between the intention to carry out the motor action and the resulting sensory consequences. The inference of this finding is profound. It implies that the internal state’s properties can affect the external behavioral performance especially in changing environment, and predictable internal state dynamics could be a necessarily condition for intelligent agents in the evolutionary pathway to have authorship of actions and to adapt themselves to changing environment.

However, there is a missing link between the findings from the simulation studies and the brain. The findings from the simulation studies do not necessarily mean that the internal state dynamics affects the external behavior (authorship of actions) in the biological brain. To fill this gap, I propose to investigate the role of predictability of internal state dynamics in the brain by analyzing the human EEG data with conscious state as a surrogate of authorship of actions.

Generally REM sleep is considered more conscious state than SWS. One approach (or example) to explain the conscious states of REM sleep and SWS is whether reporting dream or not during the different states. People usually cannot remember dream during SWS while they can do during REM. In [23], the participants were awakened during SWS and REM sleep and asked to report about their dream. Some participants awakened during SWS reported about their dreams. For reporting details of dream, REM reports were significantly longer than SWS reports, and semantic knowledge was mentioned more frequently as a dream source for REM than for SWS dream report. With these experimental reports, we can say SWS is at least less conscious state than REM.

In this chapter, we analyzed public electroencephalogram (EEG) data from the PhysioBank [43] to test our hypothesis. We took the EEG data taken during awake and sleeping states (both slow-wave sleep (SWS) and rapid eye movement (REM) sleep) and measured their predictability. Our results turned out to be consistent with our hypothesis that conscious states (awake or REM sleep) have high predictability while unconscious (or less conscious) states (SWS) have low predictability, i.e., awake and REM sleep EEG data exhibited high predictability while SWS EEG data showed low predictability.

In the following, we will present our EEG data analysis method and the results, and discuss limitations and implications of our findings on time perception and neurorobotics.

3.2 Materials and Methods

3.2.1 EEG data

For our analysis, we used the EEG data from PhysioBank. PhysioBank is a free online database that has a large, growing collection of digitized physiological signals and related data for the biomedical research community [43].

The particular database we used is the Sleep-EDF Database that is the recordings obtained from Caucasian males and females (21 to 35 years old) under no medication. The recordings contain horizontal EOG, Fpz-Cz, and PzOz EEG, sampled at 100Hz. The details of the Sleep-EEG Database is described in [62].

Among these data sets, we used the Fpz-Cz EEG data of the four subjects (two males and two females) from the database. An EEG amplifier measures voltage difference between different points on the scalp. The Fpz-Cz EEG is the measure of the two electrodes that are located at the Fpz (above the nasion) and the Cz position (top of the head), respectively.

3.2.2 EEG data analysis

Figure 3.2 shows the EEG data sets we used for our analysis, from [62]. We used EEG signals from four subjects with recordings during awake state (A–B), REM sleep (C–D), and

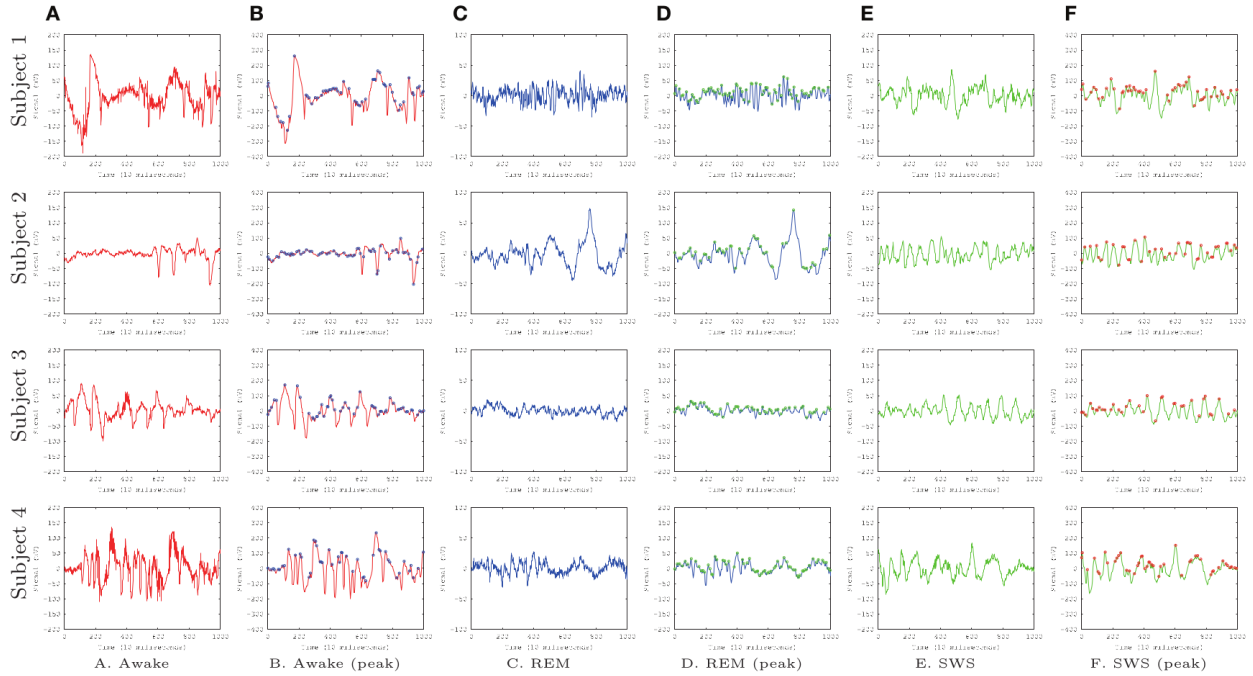


Figure 3.2: EEG data [62] from the PhysioBank [43] are shown. Each row represent data from each subject (four total) and each column represent different states. A. Awake, raw data. B. Awake, smoothed (Gaussian filter, $\sigma = 1$) and peaks identified (circles). C. REM, raw data. D. REM, smoothed and peaks identified. E. SWS, raw data. F. SWS, smoothed and peaks identified. Each data set had 30,000 data points but here we are showing only the first 1,000 for a better view of the details. Adapted from our recent work [151].

SWS (E-F). We convolved the EEG signal with a Gaussian filter with $\sigma = 1$ to smooth the signals. This was done to avoid sharp, high frequency peaks that made prediction difficult in all conditions (awake, REM, and SWS).

A perceptron (a single unit in neural networks) is a linear predictor, which is similar to the autoregressive (AR) model mentioned in [17]. Multi layer neural networks consisting of many perceptrons are nonlinear predictor. [99] used feedforward neural networks (NNs) with time delay inputs to predict nonlinear complex time series. Their neural networks predictor showed accurate predictions for the small prediction steps (less than 5 steps). This result is similar to our pilot experiment where we used feedforward NNs with time delay inputs to perform one step prediction. With the original EEG data, the prediction was pretty accurate for all REM, AWAKE, and SWS, and showed no difference between them.

Also, [32], compared the performance of the one-step-ahead prediction for EEG time series data between NN-based time series prediction and linear adaptive autoregressive (AAR) models. They presented that the NN-based time series approach is better than the linear AAR model in their experiment.

We used the feed-forward neural network predictor approach we first used in [68]; note that exact error values were used in this study instead of using the adaptive error rates. The idea is to train a feed-forward neural network using the back propagation algorithm, where the inputs are k past data points ($k = 10$ in our case) and the target output is the current data point (Figure 3.3). Each EEG time series was traversed with a window of size 10 to construct the input set and the value immediately following the time window was used as the target value, thus forming the data set. Of the full data thus generated, 60% were used for training, 15% for validation, and 25% were for testing.

Initially, we applied the above approach to predict the EEG time series directly. However, we were not able to find any significant difference in predictability across the three different conditions. Based on our successful pilot analysis of single neuron recording (spike train)

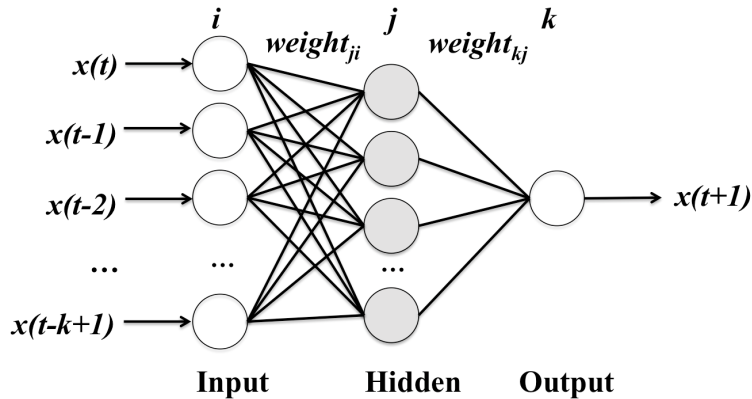


Figure 3.3: A neural network predictor for a time series. The network predict a value x at time $t + 1$ by looking k previous time steps from the current one at time t . The neurons (perceptrons) are interconnected, and the weights of the connections are adjusted to minimize the errors between the predicted output values and the ones in the train sets during training phase. We used $k=10$ past data points as inputs, 10 hidden units, and 1 output unit. The network was trained using the Levenberg-Marquardt algorithm, following [50]. In the algorithm, a damping parameter determines how much the algorithm will approximate Newton's method (small value) or gradient descent (large value). The parameter was initially set to 0.001 and its decrease factor set to 0.1 and increase factor set to 10. Training was stopped when the validation set error failed to decrease for 5 consecutive epochs or if the gradient value fell below 10^{-10} . Note, however, that the particular type of algorithm used for the prediction is not of central importance and we expect similar results with any other reasonable algorithm.

data, where we were able to predict the inter-spike interval (ISI), we considered detecting the EEG signal peaks and predicting the inter-peak interval (IPI), or inter-peak distance ([131], p. 83). Please refer to the Discussion section (Section 3.4) for more information regarding neuronal ISI prediction and why we did not include those results here.

We used a simple local search (whether data point at t has a higher value than its immediate neighbors at $t - 1$ and $t + 1$) to detect the local peaks in the convolved EEG data (Figure 3.2B, D, and F, marked with circles). From these peak locations, we calculated the inter-peak interval (IPI). Using the same neural network predictor described in Figure 3.3, we attempted to predict the $k + 1$ -th IPI, given the past k IPI values from the EEG time series. The training set was used for training and the validation set used to determine when to stop the training. The test set (novel input) was used to calculate the IPI prediction errors, by subtracting the predicted IPI from the true IPI.

3.3 Results

The IPI prediction error on novel data (not used during training or validation) are summarized in Figure 3.4 and the error distributions shown in Figure 3.5.

The results show that, for all four subjects, on average, both awake state and REM have lower IPI prediction error than SWS (Figure 3.4, all differences (except REM vs. AWAKE for the subject 4) were significant [t-test, $p < 10^{-6}$, $n \sim 2,000$]). These results support our hypothesis regarding the predictability of internal state dynamics and consciousness (i.e., they should be correlated).

Curiously, REM data had the lowest IPI prediction error, even compared to the awake state. This was somewhat unexpected since we hypothesized predictability will be correlated with the degree of consciousness and by default we expect that the awake state is the most conscious. This is an interesting counterintuitive result and we will discuss why this could be the case in the Discussion section.

The IPI prediction error distributions (Figure 3.5: note that the y -axis is in log scale) also shed some light on the properties of the predictable dynamics in the EEG data and the differences among the three conditions, awake, REM, and SWS. As expected, the IPI prediction error distributions have a higher peak near zero for awake (red) and REM (blue), compared to that of SWS (green). SWS has a much heavier tail, indicating that extreme error values are much more common in this condition. Again, these results support our hypothesis.

We also ran the FFT power spectral analysis with the EEG data to see a possibility that the different IPI prediction errors between the three conditions are just simple reflection of the power of alpha waves. Alpha waves are in the frequency range of 7.5-12.5Hz and known for synchronous, coherent, and probably most predictable neural oscillations in EEG brain signals. In the results of the FFT power spectral analysis with EEG data, alpha waves are not notably observed for all data, even in the awake states. This is partly because alpha waves are reduced with open eyes, drowsiness, and sleep. Note that the participants were doing normal activity at home with open eyes when the AWAKE EEG data was recorded. Therefore, it seems that there is no strong association between the IPI prediction and the alpha wave spectral power in the EEG data.

An interesting property of all error distributions is that the side with positive error has a broader spread compared to the side with negative error (i.e., the distribution is positively skewed, with skewness ranging from 0.86 to 1.69). Since the error is calculated as $error = true - predicted$, underestimation of the IPI seems more error-prone than overestimation. See the Discussion section for what factors we think contribute to this this kind of skewness.

3.4 Discussion

In this chapter, we analyzed publicly available EEG data from sleep and awake states to measure the predictability of the signals under conscious (awake and REM sleep) and uncon-

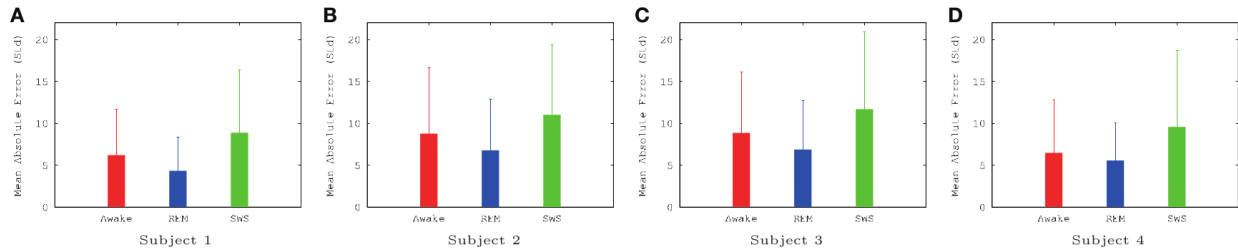


Figure 3.4: Summary of EEG IPI Prediction Error Results (Mean and Standard Deviation). Mean and standard deviation of IPI prediction error are shown for all four subjects, for all three conditions (awake, REM, and SWS). The unit for the y -axis was 10 milliseconds. For all subjects, awake and REM conditions resulted in lower IPI prediction error than SWS, showing that predictive dynamics may be more prominent during conscious states. All differences were significant (t-test, $p < 10^{-6}$), except for REM vs. AWAKE for subject 4. See text for details. Awake state having higher IPI prediction error than REM state is somewhat unexpected, which we will discuss further in the discussion section. Adapted from our recent work [151].

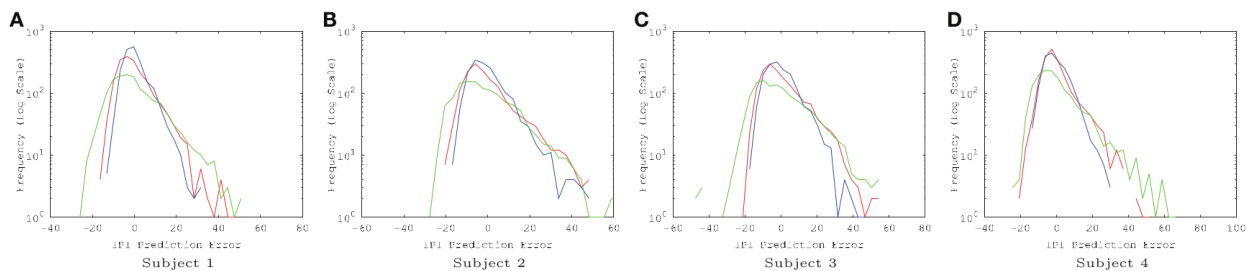


Figure 3.5: EEG IPI Prediction Error Distribution The IPI prediction error distribution is shown for all four subjects, each for all three conditions (awake [red], REM [blue], and SWS [green]). The x -axis is in linear scale while the y -axis is in log scale for a clearer view of the probability of extreme error values. The unit for the x -axis was 10 milliseconds. The trends are consistent for all four subjects. REM has the highest peak near zero error, closely followed by awake state, and finally SWS which shows the lowest peak. SWS has the heaviest tail, meaning that high error values are much more common than awake state or REM. Adapted from our recent work [151].

scious (SWS) conditions. We found that the predictability of EEG signals correlated with the degree of consciousness. These results support our earlier hypothesis that predictable internal brain dynamics is a necessary condition of consciousness. In the following, we will discuss potential issues and interesting observations from our study, and propose potential applications of our finding to time perception and neurorobotics.

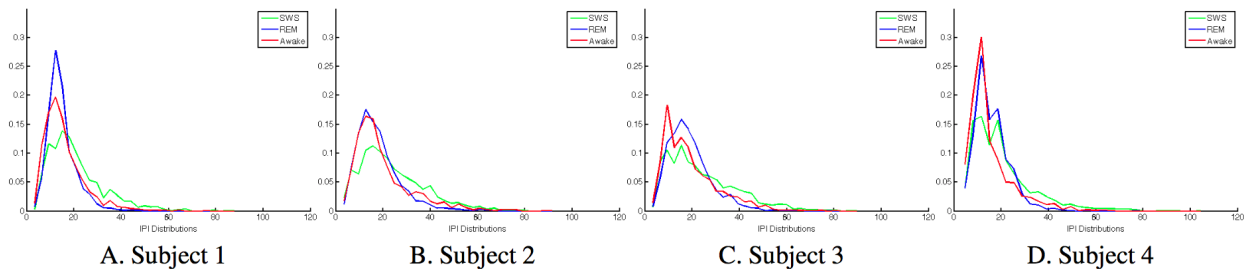
There are potential limitations of our approach as we briefly mentioned in the Materials and Methods section (Section 3.2). First, we measured predictability in the inter-peak interval in the EEG signals, not directly on the raw EEG signals. Predictability measured on raw EEG signals did not show any significant differences among the three conditions: awake, REM, and SWS (pilot results, data not shown). This could be due to multiple factors, one of which is the nature of the EEG signals. For example, EEG signals are weighted mixtures of on-going electrical activity in the brain. Also, generally reduced levels of activity during SWS may result in flatter signals (slowly changing and low-amplitude, further confounded by mixing) which may be easier to extrapolate from. Based on this observation, we initially analyzed single neuron spike train data obtained during sleep and awake states by [119]. Using the data, we used the same feed-forward neural network predictor to predict the inter-spike interval (ISI) under awake, REM, and SWS conditions. Our results were consistent with what we reported here, however, the data set was very small (on the order of 100 spikes per condition, compared to thousand peaks in the EEG data) so we could not draw meaningful conclusions. However, since we found that using discrete events (spikes) instead of the continuous wave form gave promising results, we tried to recover such events in the EEG data which led us to the inter-peak interval (IPI) measure; (however, note that we are not extracting spike information from the IPI information).

One rather unexpected result was that the IPI prediction error was lower for REM sleep than awake state, and significantly so (t-test, $p < 10^{-6}$ in all cases (except REM vs. AWAKE for the subject 4)). Does this mean that subjects are more conscious during REM sleep than

when they are awake? The reason for this may again be due to the mixed nature of EEG signals, plus the natural sources of randomness in the stimulus environment during the awake state. Because the awake EEG signals are driven both by the internal brain dynamics and the external stimuli, a mixture of the two may be slightly less predictable. A possible way to isolate the internal vs. external sources would be to use blind source separation, e.g., independent components analysis [33], and correlate the isolated components with the stimulus statistics. This way, we can rule out the externally driven signal variability during awake state. Our prediction is that the predictability of these internal components would be as high as that of the REM data.

Another interesting property of the IPI prediction error distribution is its positive skewness under all conditions. Positive skewness means more positive error than negative error, which indicates underestimation of IPI (since $error = true - predicted$). One possible explanation for this is that the prediction mechanism may be tuned more to shorter IPIs as the EEG signals generally tend to show high-frequency bursts followed by occasional pause of low-frequency intervals. The IPI distribution (Figure 3.6) shows that, for all cases, the distributions are positively skewed, and so the number of IPI instances smaller than the means is less than the others (the number of instances bigger than the means). This trend can explain the positive skewness of the IPI prediction error.

The methodology and results from this chapter can also contribute to a better understanding of higher level cognition related to time perception. In a general sense, our findings suggest an important link between subjective mental phenomena and time, supporting recent results that relates affective states and the experience of time (see [143] for a review). Also, longer-time-scale events signified by IPI may have closer ties to higher level cognitive function. [74] showed that in a computational simulation that in a bottle-necked Continuous Time Recurrent Neural Networks in a cognitive timed decision task, higher parts of the bottle-neck evolve slower dynamics than the lower parts in the network. Such a multi-scale,



Conditions	Subject 1	Subject 2	Subject 3	Subject 4
SWS	1.2108	0.8583	0.8293	1.6584
REM	2.3728	1.5487	1.3675	2.1851
AWAKE	1.7577	1.6598	1.6238	2.7095

E. Skewness

Figure 3.6: EEG IPI Distribution A-D. The IPI distributions are shown for all four subjects, each for all three conditions (awake [red], REM [blue], and SWS [green]). For all cases, the IPI distributions are positively skewed. E. The skewness values are in the table.

multi-mechanism aspect of the experience of time could also be linked to different classes of movements, e.g., discrete vs. continuous movement classes [57]. As suggested by [36], development of temporal processing capabilities may be dependent on a rich set of underlying internal brain dynamics and studies like ours can help understand the requirements and mechanisms of temporal processing in the brain. Our results showing that underestimation of IPI is more error-prone than overestimation is also suggestive, i.e., overestimation may have less severe consequences on the organism. [87]’s results showing human’s tendency to overestimate time duration (in an embodied context) is in line with our observation. Investigating such biases (underestimation vs. overestimation) can lead to new insights on time perception.

Finally, findings from our study can provide new directions for neurorobotics research with a focus on time perception and temporal processing. Again, the use of IPI as a basis of temporal event prediction can be adopted by neurorobotics, e.g., in reward event prediction

[134]. Instead of continuous estimates of future event timing, discrete rank ordering could be employed (e.g., analogous to rank ordering in visual depth perception). This could be especially interesting if predictions of two different temporal events need to be compared. Furthermore, multi-scale aspect can be introduced by convolving the robot's internal brain dynamics with Gaussian kernels with a different width, thus allowing the robot's higher level temporal processing mechanisms to utilize variously timed events. An open research problem with respect to the above is how predictable internal dynamics can lead to the actual use of the dynamics in prediction tasks that are behaviorally relevant to the robot, especially involving motor tasks such as catching a falling ball [80]. Another interesting question is to ask what are the origins of predictable dynamics. Our earlier studies showed that delay and predictive capabilities are intricately tied together: Paradoxically, increasing delay (e.g., axonal conduction delay) in the system tended to lead to more predictable dynamics [73, 75]. This could be used as a round-about way for developing predictive capabilities in neurorobotics, by a strategic use of delay intentionally planted in the system. Also, a general lesson may be that delays are not always bad (see, e.g., [127] where broader delay distributions led to the abolishment of complex dynamics in a neural network). A deeper understanding of this connection can lead to robust time perception and control mechanisms in intelligent sensorimotor agents.

4. EMERGENCE OF TOOL USE BY EVOLVED NEURAL CIRCUITS

4.1 Introduction

Tool-use [52], which requires high levels of sensorimotor skill learning and problem solving capabilities, is one of the salient indicators of intelligence along with communication (language) [117] and logic [4]. Communication and logical inference have been extensively investigated in artificial intelligence (AI). However, tool-use is still largely under-developed in the field of AI [27].

In this chapter, I investigate how the capability to use tools can spontaneously emerge in an evolved neural circuit controller for a two degree-of-freedom articulated limb. The goals of the project in this chapter are to find minimal and indirect fitness criteria for the emergence of tool-use, and to analyze the properties of evolved neural circuits that permit tool-use.

To support the background and motivation in detail, the rest of this section consists of the review of tool-use in animals and in AI/Robotics, followed by the approach used in this project.

4.1.1 Tool Use in Animals

Tool construction and use require high levels of sensorimotor skill, planning, and problem solving capabilities. Only a small number of animals exhibit the capability of tool-use and yet fewer are known to construct tools (see [114] for an overview).

Tool use is observed in various animals, not just in humans. For example, chimpanzees use primitive tools such as stones or sticks [19, 138], macaque monkeys use stone axes for

Part of this section is reprinted with permission from Qinbo Li, Jaewook Yoo, and Yoonsuck Choe, "Emergence of tool use in an articulated limb controlled by evolved neural circuits." In Neural Networks (IJCNN), 2015 International Joint Conference on, pages 1983-1990. IEEE, 2015. c 2015 IEEE.

various purposes [49], parrots use sticks to reach objects beyond a barrier [9], Degus (a South American rodent species) can be trained to use a rake-like tool to obtain food under a fence [89], elephants also use sticks or other objects to obtain food located out of normal reach [40], and dolphins use sponges to cover their snout when digging for food [116].

Tool construction is only rarely observed in non-human animals. Chimpanzees have long been known to construct simple tools such as a stack of boxes to reach high-hanging fruit [64]. More recently, Price et al. [98] showed that Chimps, after training, can put together two sticks to extend the reach of the tool. Latest results also show that wild monkeys flake stone tools to obtain sharp edges [100]. Non-primate species have also shown tool construction capability, although limited in its complexity. For example, New Caledonian crows in captivity have been observed to bend a wire to create a hook to retrieve food [63]. However, tool construction (especially in the wild) is extremely rare among non-human animals.

There are several levels of tool construction and use with increasing difficulty [27]. Level 1 corresponds to the use of unmodified objects as a tool, which is most commonly observed in animals. Level 2 represents the construction of simple tools through modification of an object or putting together a small number of objects. Level 3 is where multiple objects are put together to form a more complex tool. Level 4 involves two or more agents cooperating to construct a tool, including abstract (or social) tools. Level 5 calls for the ability to explain how tools are constructed, used, and why. Animals other than humans have only reached level 2. In this dissertation, I will focus on task up to level 2 since this is already a very challenging task. Note that even this early level has not been explored in AI research (Table 4.1)

4.1.2 Tool Use in AI and Robotics

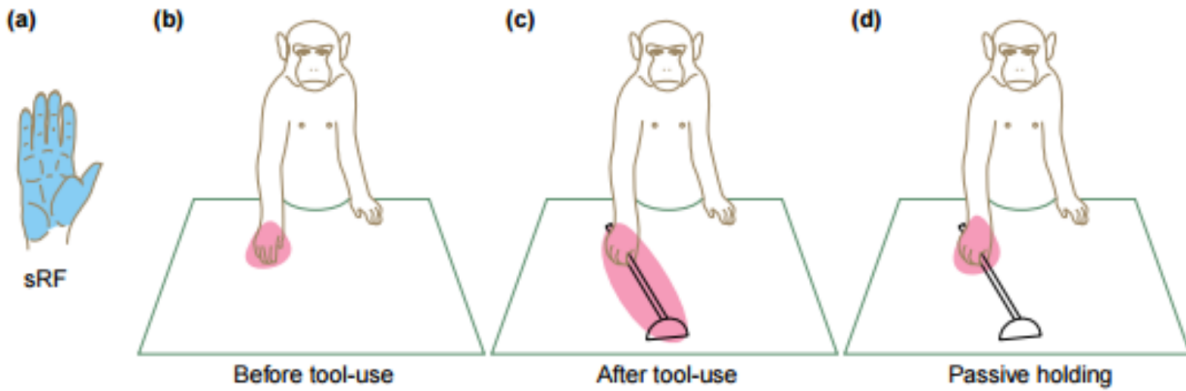
Tool use [52], which requires high levels of sensorimotor skill learning and problem solving capabilities, is one of the salient indicators of intelligence along with communication (language) [117] and logic [4]. Communication and logical inference have been extensively investigated in artificial intelligence (AI). However, tool-use is still largely under-developed in the field of AI [27].

4.1.2.1 Related work

In artificial intelligence and robotics, tool-use has recently gained attention (See [5]), however tool construction is a relatively unexplored area. For a review, the various existing works are listed as follows. See Table 4.1 for a comparison, where six features of the existing studies are listed. Among the features, the meaning of “Affordance” is to recognize “Where is the tool handle”, “What function does the tool afford?”, etc. [120, 128]. “Tool-body assimilation” means “How does ones body image change or extend when the tool is used?” ([78][88], Fig. 4.1). Studies with “Real Robots” and/or “Simul Physics” include real robots and/or physics simulation in experiments.

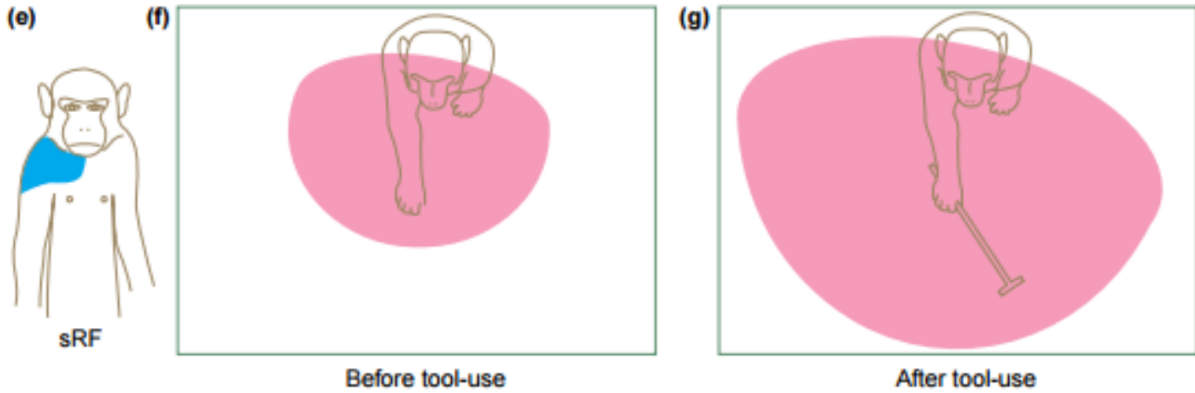
- (1) Wood and Amant [147] used state-machine-based behavior using maximum-margin clustering, where a robot dog was used for a reaching task with a stick. The set of states and actions was pre-programmed.
- (2) Learning through human demonstrations has been studied in [7, 71, 93, 108, 148]. In [7] by Arsenio, a humanoid robot learned to use tools such as hammers, drums, saws, brushes, belles, etc. from human demonstrations. Tasks were modelled as a finite Markov Decision Process (MDP) including states, tool-use actions (e.g. hammering, dropping, poking, etc.), controller of the robot, and so on. In [71] by Lee et al., tool-use motion knowledge (both tool manipulation knowledge and body motion

Distal-type neurons



Distal Receptive Field (hand)

Proximal-type neurons



Proximal Receptive Field (shoulder)

Figure 4.1: Tool-body assimilation (Receptive Field Changes Due to Tool Use). The receptive fields for distal (end effector: hand) and proximal (first joint: shoulder) parts of the arm of bimodal (visual [pink] and somatosensory [blue]) neurons are shown. The visual RFs of these neurons are known to extend due to tool-use: (c) and (g). Adapted from [78].

knowledge) was learned from human tool-use demonstrations. Examples of tool-use in this study are wielding a tennis racket, a wooden sword, a golf club, etc. Both a simulated physics and a humanoid robot were used for this work. Pastor et al.[93] had humans physically guide the robot arm to generate demonstrations for tool-use movement trajectories. The task included grasping and placing an object (e.g. cup), where a seven degree-of-freedom (DOF) robot arm was used for the experiment. In [108], grasping, holding, and dropping (a block) actions were sequentially presented by human. Based on the demonstrations, a humanoid robot reproduced the actions by mapping the corresponding sensory feedback from the observations to the robot's own sensory feedback. In the study by Wu and Demiris [148], humans demonstrated how to use five different labelled tools (such as spanner, hammer, knife, etc.), where the agent learned to classify the kind of tools by observing the tool-use movement from the demonstrators (humans). The dynamic properties of the tools were observed. Affordance, was used to classify tools, and hierarchical representations of the tools were used for the learning.

- (3) In [21] by Brown and Boesch, planning through inductive logic programming such as go-to, pickup-object and drop-object, and pull-with-tool actions was used for tool-use actions. (Also see [90], where a model of inductive logic programming was proposed for tool-use actions.)
- (4) Learning affordances via random trial-and-error or body babbling was investigated in [22][60][120]. In Bullock et al.[22], a three DOF arm learned to reach a distant target with tools (sticks) of variable lengths. The tool-use was learned through motor babblings of the arm. In [60] by Katz and Brock, the task was learning kinematic model of tools (e.g. scissors, shears, pliers, a stapler, etc.) by interacting with the tools based on pre-programmed motions. A seven DOF robot arm was used. In the

study by Stoytchev[120], a five DOF robot arm learned affordance of tools through behavior babbling. The behaviors were encoded manually by the researchers. A stick, L-stick, T-stick, L-hook, and T-hook are examples of the tools.

- (5) Tool use based on tool-body assimilation was studied in [88][126]. Nishide et al.[88] presented a method to apply a robot’s active sensing experience to create tool-body assimilation model. The model learned that the robot’s dynamical properties change when holding a tool. The tools were L-shaped, T-shaped, and I-shaped sticks. The tools were pre-attached to the robot arm while generating nine pre-programmed motions for the learning. The task was moving objects to target locations. A seven DOF arm of humanoid robot was used. Takahshi et al.[126] proposed a tool-body assimilation model, where a recurrent neural network with multiple time-scales was used for the body model learning. For exploration of the tool functions, predefined motions were used. I-shaped tool, T-shaped tool, and L-shaped tool were used. A physic simulator was used, where a robot arm was simulated to manipulate simple object moving tasks.
- (6) Bayesian learning of tool affordances was investigated by Jain and Inamura [58]. The tool affordance (that is the surface on which the tool contacts the object) was learned by Bayesian learning. The Bayesian networks representing tools were updated by interacting with humans. Various shapes of sticks and rakes were used as tools. The experiment was conducted in a physic simulator. In [45] by Goncalves et al., Bayesian network learning was applied for learning affordance of tools (e.g. stick, L-stick, fork, etc.) and objects (e.g. ball, cube, etc.) as well. Four directional pushes were defined as actions (e.g. left, right, pull closer, and push away). For the effects of actions, five discrete levels were defined. Both a physic simulator and humanoid robot were used.
- (7) Tikhanoff et al.[129] investigated affordance learning with least square support vector machine. Rolling and pulling affordance was learned. The rolling affordance repre-

sented the property of the object (such a toy car) to react (to roll a certain distance) to a tapping action from the robot. The pulling affordance meant the property of the tool (such as a rake or a stick) to pull an out-of-reach object, where how the tool gets in contact with the object for pulling needs to be learned. A humanoid robot was used for this work.

- (8) In [121] by Stuckler and Behnke, tool-use was investigated in a service robot (with an anthropomorphic upper body). They proposed to adapt the tools themselves to have special handles for providing stable grips for the robot. RGB based perception algorithm was applied to segment objects. Various tool-use strategies such as sweeping-up-dust, bottle-opening, and watering-plants were pre-programmed.
- (9) Self-supervised learning of grasp-dependent tool affordance was investigated by Mar et al.[77]. Learning pull affordance of tools were conducted by considering both how the tool is grasped and how the action is performed. Three different orientations were used for grasping the tools (e.g. rake, stick, L-stick, etc.). The effects of tool-use actions were measured as the displacement of a cube object. The affordances were discovered by clustering the observed effects. Both a physics simulator and a humanoid robot were used.
- (10) In [84] by Mokom and Kobti, tool capabilities (affordances) recognition task was conducted. Tool types and their capabilities were encoded as IDs, and a genetic algorithm was used for the tool capability recognition task in a simple grid world. A tool capability represented a way the agent can successfully use the tool.
- (11) Evolved tool using behaviors were investigated by Chung and Choe [28], where the agents were evolved with a genetic algorithm to learn to use (i.e. to drop and detect) markers for a food foraging task or a ball catching task. In Schafer and Bergfeldt's

work[111], the agents (predators) were evolved with a genetic algorithm to learn to catch an evasive prey agent effectively using tools in a grid world. Two types of tools were used. When one of the tools was acquired, the agent’s sensory or motor ability was improved.

4.1.2.2 Two Gaps

There are two gaps in tool construction and use in AI and Robotics: (1) Most of the works listed above depend on some degree of designer knowledge regarding tool-use and motor control, e.g., fully hard coded behavior, the tool being pre-attached to the limb, pre-defined tool features, pre-defined motor primitives, etc. Evolution-based approaches [28][111] were relatively free of these constraints, but in those cases the tools were more or less simple markers, not something than can be manipulated with a limb-like structure of the agent. (2) Furthermore, AI-based work on tool construction is almost non-existent (see Table 4.1). Wang et al.[135] took a synthetic approach to construct tools, but their focus was more on understanding the various mechanistic and energetic needs of using the synthetically generated tools, not on tool construction by agents. In the recent study by Reams and Choe [101], the agents (with two DOF articulated limb) were evolved for a simple tool construction (combining two sticks) task to reach distant targets, but the task was a simple reach using one type of tool (stick). Also, the locations of the tools and targets were encoded.

4.1.3 Approach

In this chapter, we investigate how the capability to use tools can spontaneously emerge in an evolved neural circuit controller for a two degree-of-freedom articulated limb. The goals of the study were to find minimal and indirect fitness criteria for the emergence of tool-use, and to analyze the properties of evolved neural circuits that permit tool-use. We evolved the neural circuit controller by gradually augmenting the network topology (NeuroEvolution of Augmenting Topologies [NEAT], by Stanley and Miikkulainen [118]). The environment

Reference	Learning	Affordance	Tool-body Assim.	Real Robots	Simul Physics	Tool Constr.
Wood and Amant[147]				✓		
Arsenio[7]	✓	✓		✓		
Lee et al.[71]	✓	✓		✓		
Pastor et al.[93]	✓			✓		
Saegusa et al.[108]	✓			✓		
Wu and Demiris[148]	✓	✓				
Brwon and Boesch[21]		✓				
Bullock et al.[22]	✓	✓	✓			
Katz and Brock[60]	✓	✓		✓		
Stoytchev[120]	✓	✓		✓		
Nishide et al.[88]	✓	✓	✓	✓		
Takahshi et al.[126]	✓	✓	✓		✓	
Jain and Inamura[58]	✓	✓			✓	
Tikhanoff et al.[129]	✓	✓	✓	✓		
Stuckler and Behnke[121]				✓		
Goncalves et al.[45]	✓	✓		✓	✓	
Mar et al.[77]	✓	✓		✓	✓	
Mokom and Kobti[84]	✓	✓				
Chung and Choe[28]	✓					
Schafer and Bergfeldt[111]	✓					
Wang et al.[135]				✓		✓
Reams and Choe[101]	✓		✓			✓
Proposed (Chapter 4)	✓	✓	✓			✓
Proposed (Chapter 5)	✓	✓	✓		✓	✓

Table 4.1: Tool Construction and Use in AI and Robotics. Six features of the existing works listed in text are compared. The meaning of “Affordance” is to recognize “Where is the tool handle”, “What function does the tool afford?”, etc. “Tool-body assimilation” means “How does ones body image change or extend when the tool is used?” “Real Physics” and/or “Simul Physics” mean studies include real robots and/or physics simulation in experiments. See the text for details.

consisted of a two degree-of-freedom articulated limb, a target object, and a tool. The task was to reach a target object that may or may not be within reach of the limb, with or without the tool. Different fitness criteria were tested to investigate which one is good enough to give rise to tool using behavior for the limb. We define three basic fitness criteria and tested their different combinations to evolve the neural circuit for the controller for the limb. We avoided using direct heuristics such as predefined motions or programmed guidance. Our results indicate that simple, indirect criteria such as distance to target, number of steps to reach target, and if the tool was fetched was enough to give rise to tool using behavior. Quantification of recurrent loops in the neural circuit topology showed that better controllers have more such loops. Furthermore, if recurrent loops are prohibited, the task performance greatly degraded and the evolved circuit had much more neurons than their recurrent circuit counterpart.

The rest of the chapter is organized as follows. We will first describe our approach and method in Section 4.2. Details of the experiments are described in Section 4.3. Results are reported in Section 4.4, followed by discussion and future work in Section 4.5 and conclusion in Section 4.6.

4.2 Methods

We evolved neural circuits of arbitrary topology to control a two-limbed articulated arm in an object reaching task. The environment was equipped with a tool (a stick) that can be picked up and used to get to objects beyond the reach of the limb.

4.2.1 Algorithm for Evolving Neural Circuits

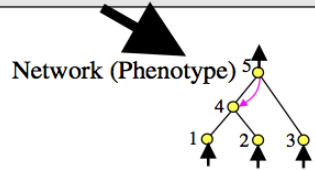
Early efforts in neuroevolution were limited to adjusting the connection weight, while leaving the neural circuit topology fixed [85, 139, 140]. In these approaches, each genotype was mapped to a full neural network. However, these approaches were not flexible enough and could not handle increasing levels of task complexity. More recent approaches were

based on single neuron-level evolution [3, 44, 86, 97], however, the evolved neurons had to be assembled into a network, which typically had a fixed topology. In this chapter, I needed a method that allows the network topology to evolve since increasingly complex tools and behavior are required, thus the methods above are not suitable. There are several approaches that allow network topology evolution [42, 150], however, these were based on weight-topology co-optimization, thus, they were not flexible enough. For this chapter, I used a neuroevolution technique called the Neuroevolution of Augmenting Topologies, or NEAT [118]. Unlike most other neuroevolution techniques, NEAT allows the neural circuit to evolve to have an arbitrary connection topology.

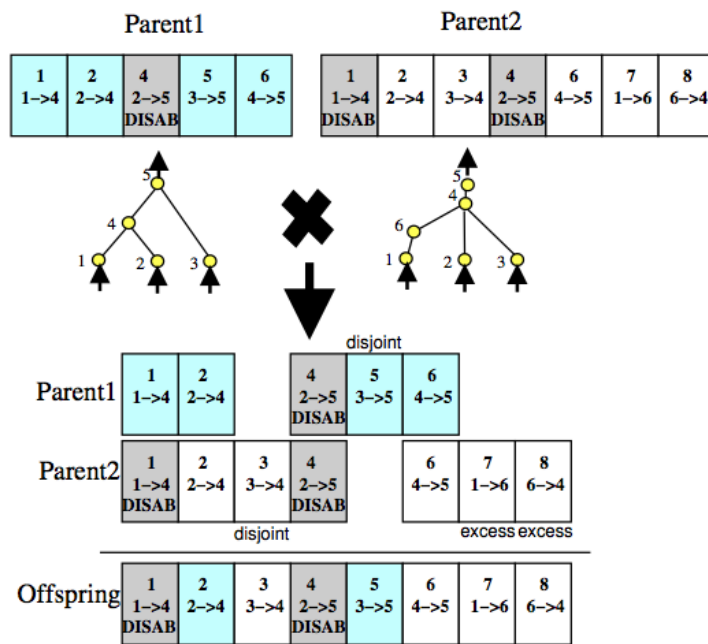
In NEAT, the chromosome encodes neurons and their connections separately, as well as the connection weights. Neurons and connections can be added or removed to change the network topology, thus the chromosome has a variable length. Mating of chromosomes with different network topology is achieved through the use of a quantity called innovation number, unique to each gene, that indicates the evolutionary origin of that particular gene. Innovation numbers allow only compatible genes to mate (i.e., genes that have the same ancestral origin).

See Fig. 4.2 above for the genotype to phenotype mapping, and crossover of topologically different networks. Mutation is not shown in the figure but its implementation is straightforward (insert or delete connections or neurons). Another unique mechanism of NEAT is speciation that helps freshly changed topology to be preserved despite the initial plunge in fitness. The rest of the algorithm is similar to other neuroevolution or evolutionary algorithms: instantiate phenotype from genotype \rightarrow test in the task environment \rightarrow calculate fitness \rightarrow selection and reproduction.

Genome (Genotype)						
Node	Node 1	Node 2	Node 3	Node 4	Node 5	
Genes	Sensor Input	Sensor Input	Sensor Input	Hidden Hidden	Hidden Output	
Connect.	In 1 Out 4	In 2 Out 4	In 2 Out 5	In 3 Out 5	In 4 Out 5	In 5 Out 4
Genes	Weight 0.7 Enabled Innov 1	Weight 0.5 Enabled Innov 3	Weight 0.5 DISAB Innov 4	Weight 0.2 Enabled Innov 5	Weight 0.4 Enabled Innov 6	Weight 0.6 Enabled Innov 10



(a) Genotype



(b) Crossover

Figure 4.2: Neuroevolution of Augmenting Topologies (NEAT). (a) The genotype-to-phenotype mapping in NEAT is shown. Each node and each connection has a gene. Each connection gene has an enable/disable flag and a unique identifier, the innovation number. (b) Crossover of two parents with different topology is shown. The genes of the two parents are aligned so that the innovation numbers match up. Adapted from [118].

4.2.2 Fitness Criteria

Our aim in selecting the fitness criteria was to find measures that do not directly dictate the evolved tool-use behavior (i.e., minimal and indirect measures). We used three basic fitness criteria and tested different combinations of them to evolve the neural circuit controller for the limb. For each controller, by the end of 100 trials, the following quantities were calculated: (1) D : distance between the end effector and the target (Eq. 4.1); (2) S : steps taken to reach the target (Eq. 4.2); and (3) T : tool pick-up frequency (Eq. 4.3).

$$D = 1 - \frac{\sum_k \|\vec{o}_k - \vec{e}_k\|}{K \times D_{max}}, \quad (4.1)$$

$$S = 1 - \frac{\sum_k s_k}{K \times S_{max}}, \quad (4.2)$$

$$T = \frac{\sum_k t_k}{K}, \quad t_k = \begin{cases} 1 & \text{if tool was picked up} \\ 0 & \text{otherwise} \end{cases} \quad (4.3)$$

where \vec{o}_k and \vec{e}_k are the coordinates of the target object and the end effector of the limb, respectively. The Euclidean distance $\|\cdot\|$ is normalized by the maximum radius of the environment D_{max} . K indicates the total number of the trials ($K = 100$ in the experiment) and k indicates k_{th} trial. s_k indicates the number of steps taken before reaching the target, and S_{max} is the maximum movement steps for each trial ($S_{max} = 500$ in the experiment). t_k is 1 when the tool was picked up during the trial and 0 otherwise. Different combinations of the fitness criteria elements were tested: D , S , DS , DT , ST , and DST . Multiplication (“ \times ”) was used when combining the fitness criteria elements (e.g., DS means $D \times S$). In addition, neural circuits evolved with and without recurrent connections were compared ($SST = S^2T$ and $SST_{noRecur} = S^2T_{noRecur}$).

4.2.3 Input and Output Interface for the Neural Circuit

It is important to define the right sensory inputs to represent the environment properly, especially for artificial agents learning through action and feedback loops [137]. It is widely believed that we encode the space around us in a body-centered coordinate system. In [76], the authors compared two forms of representations: world-centered sensory representation (WC) and agent-centered sensory representation (AC). For example, AC can use polar coordinates while WC can use Cartesian coordinates. The authors further proposed the relative agent-centered sensory representation (RAC), which is the relative difference between two ACs.

In our experiment, we used RAC as the sensory inputs to the limb controller neural circuit. To be specific, the RAC between end effector and target, and the RAC between end effector and the tool were used. The details for AC (Fig. 4.3) and RAC are as follows.

$$AC_{end_eff} = (\varphi_1, d_1) \quad (4.4)$$

$$AC_{target} = (\varphi_2, d_2) \quad (4.5)$$

$$AC_{tool} = (\varphi_3, d_3) \quad (4.6)$$

$$RAC_{end_eff\&target} = (\varphi_2 - \varphi_1, d_2 - d_1) \quad (4.7)$$

$$RAC_{end_eff\&tool} = (\varphi_3 - \varphi_1, d_3 - d_1) \quad (4.8)$$

Other inputs for the limb controller were the joint limit detectors $(\vartheta_1, \vartheta_2)$ that can sense whether the joints reach the limitations as follows. The last two inputs were the current

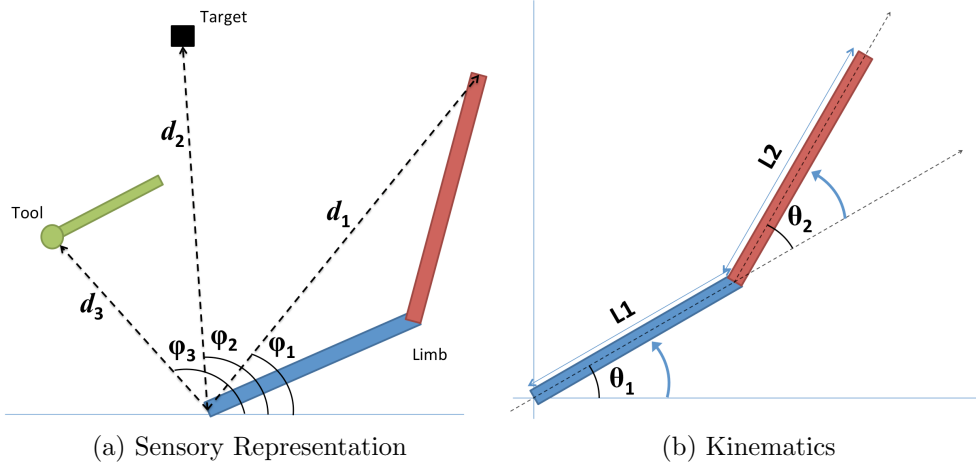


Figure 4.3: Agent-centered sensory representation (AC) and kinematics of the joint arm. (a) AC uses a polar coordinate system. φ_1, φ_2 , and φ_3 represent the angles for the end effector, the target, and the tool, respectively. d_1, d_2 , and d_3 indicate the distances to the end effector, the target, and the tool, respectively. (b) The limb consists of two joints J1 (θ_1) : J2 (θ_2), and the arm segments $L1 : L2$ (with the length ratio of $L1 : L2 = 1 : 1.25$). The two joint angles $\theta_1 : \theta_2$ are controlled by the neural circuit output.

angle of the two joints (θ_1, θ_2).

$$v_{\{\theta_1, \theta_2\}} = \begin{cases} 1 & \text{if } \{\theta_1, \theta_2\} \geq 150^\circ \\ 1 & \text{if } \{\theta_1, \theta_2\} \leq -150^\circ \\ 0 & \text{otherwise} \end{cases} \quad (4.9)$$

In a nutshell, each neural circuit consists of eight inputs, two outputs, zero or more hidden neurons, and positive (excitatory) and negative (inhibitory) connections (Fig. 4.4). The eight sensory inputs are RAC between the end effector and the target, RAC between the end effector and the tool, the two joints angles (θ_1, θ_2), and two limit detectors for θ_1 and θ_2 . The input values are normalized between 0 and 2 except the d_1 , and d_2 for RAC.

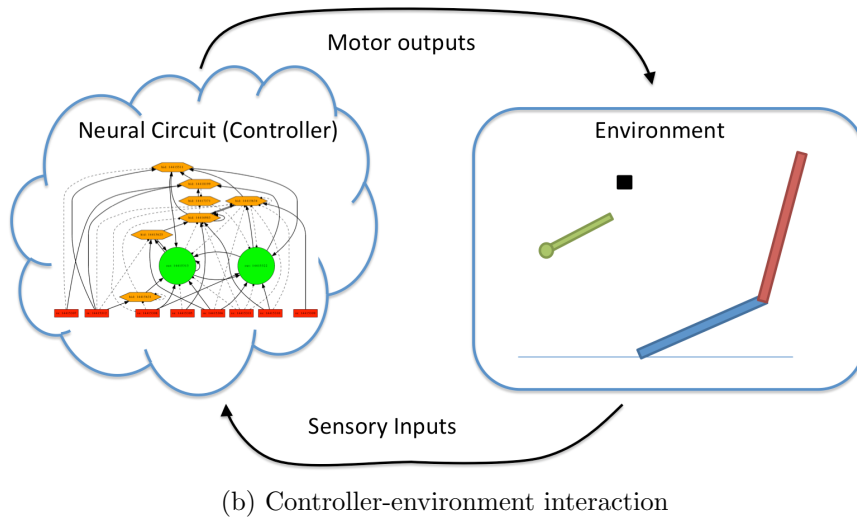
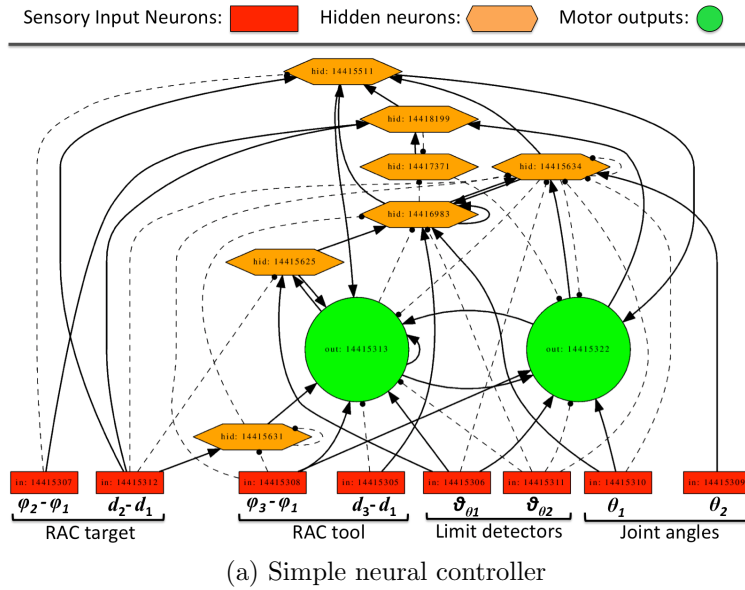


Figure 4.4: Neural circuit controller and the jointed limb. (a) A neural circuit controller consists of eight sensory inputs, two motor outputs, and zero or more hidden neurons. The sensory inputs are RAC between the end effector and the target, RAC between the end effector and the tool, two joints angles (θ_1 , θ_2), and two limit detectors for θ_1 and θ_2 . The motor outputs are for the two joint angles. The neurons are connected with excitatory (solid) and inhibitory connections (dashed) including recurrent connections. (b) The interaction cycle between the neural circuit controller and the environment is shown. When the end-effector reaches the tool end, the tool is automatically attached to the limb and the end of the tool becomes the end-effector. Note: the controller was not explicitly notified of the limb extension due to tool pick-up.

4.3 Experiment (Simulation)

The task environment consisted of a two degree-of-freedom articulated limb, a target object, and a tool (Fig. 4.4b and Fig. 4.5). The articulated limb moves on the 2D plane by changing θ_1 and θ_2 . The task is to reach the target with or without the tool. The details about the limb, the task, and the experimental procedure are described below.

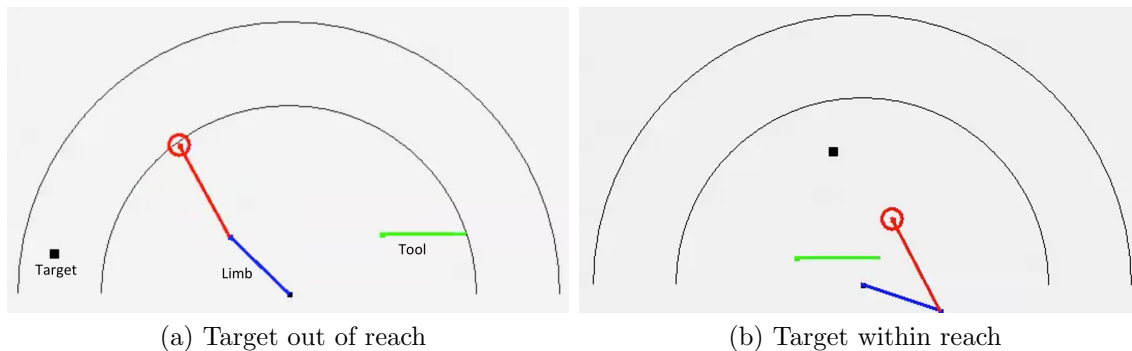


Figure 4.5: Example task conditions. The environment consists of two degree-of-freedom articulated limb, a target object, and a tool, all on a 2D plane. Note that the tool’s initial locations are variable. The two half-circles indicate the original reach (inner) and the reach with tool-use (outer).

4.3.1 Reaching Task

The task is to evolve a neural circuit controller to reach targets with or without the tool. Amant and Wood [5] argued that animals such as chimpanzees, elephants, and parrots have some level of intelligence that can be measured, and similar experiments and metrics can be applied to artificial agents. In [78], a task for reaching a distant object using a tool (stick) was used to observe tool-use in monkeys. They found that the body schema changes in the monkey’s brain when the tool was in use. The task of reaching a distant object using a tool can be categorized as simple tool-use, according to the “tooling test” taxonomy proposed by

Amant and Wood [5].

4.3.2 *Articulated Limb*

The articulated limb consisted of two joints (θ_1 and θ_2), and two links ($L1$ and $L2$) (Fig. 4.3). The limb was able to move on the 2D plane by changing θ_1 and θ_2 . It can move the two joints left or right. At each time step, the velocity limit was from -1.5° to 1.5° . The two outputs of the neural circuit (the limb controller) were connected to the two joints to move the limb. The ranges of the two joints were from -150° to 150° .

4.3.3 *Experimental (Simulation) Procedure*

Each two degree-of-freedom articulated limb was given 100 trials to perform the target reaching task. In each trial, the limb was allowed to move up to 500 time steps (maximum time step). For each trial, the target and the tool were placed at random locations. The distance of the target could be either within the reach of the limb or not (the tool was always within the reach of the limb; see Fig. 4.5). In the latter case, the limb must use the tool to reach the target (otherwise it will fail). If the limb successfully reached the target, the current trial ended and the number of time steps taken was recorded. If the agent failed to reach the target, the distance from the end effector to the target was recorded. The fitness of the agent was calculated based on the sum of final distances (Eq. 4.1), the total time steps (Eq. 4.2), and the number of times the tool was picked up (Eq. 4.3), as described in the Methods section. Different combinations of the fitness criteria elements were tested as described in Section 4.2.

We evolved the the neural circuit (the limb controller) for 120 generations with a population size of 100. To evaluate the performance, after the 120 generations were done, we picked the best limb controller and repeated 10 times a set of 100 test trials (total 1,000 trials). With this, we compared the performance and the trends throughout the generations, as well as the network topology characteristics. For all experiments, the within-reach and

out-of-reach conditions were balanced (50% each).

4.4 Results



4.4.1 Evolved Neural Circuits and Observed Behavior

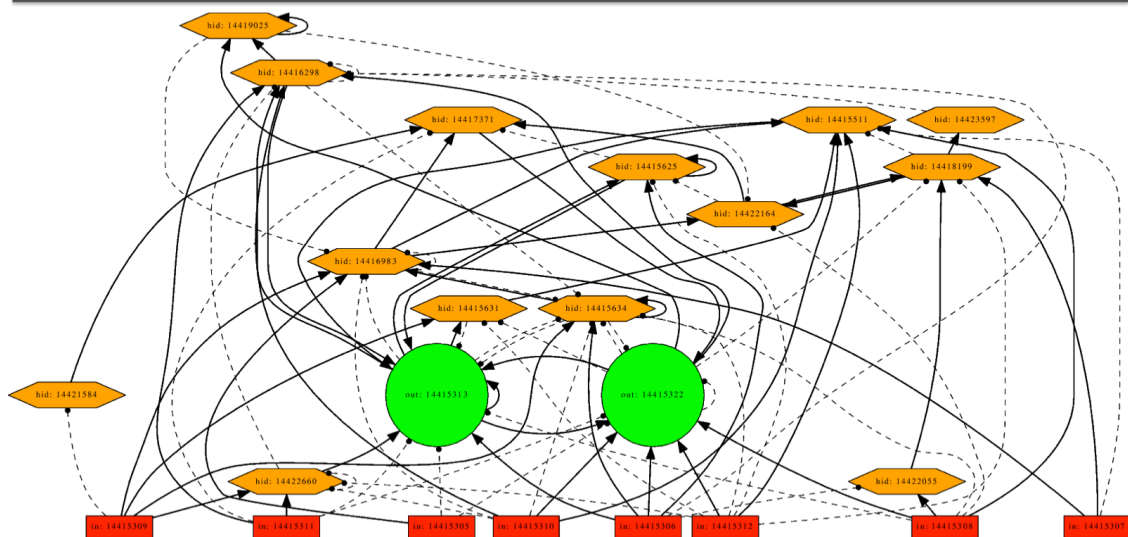
Fig. 4.6 shows the network topology of evolved circuits under different fitness criteria. The one that included tool pickup frequency (T) in its fitness showed a much simpler network while maintaining the same level of performance. In Fig. 4.7, limb movements are shown as time-lapse images. Time is encoded as the pixel intensity, where darker means more recent state. The three images in the top row show examples of successful movements. Starting position is marked as \textcircled{S} , tool pick up event as \textcircled{T} , and ending position as \textcircled{E} . Target location is marked \square . The limbs (blue and red lines) in the three images first move towards the tool (green line with a circle at the handle) to pick it up, and then the extended limb moves towards the target. The trajectories of the end effectors are shown as black solid curves. In the bottom row, the three images show examples of unsuccessful movements, where the three movements of the limb could not reach the target (\textcircled{E} is not near \square).

In successful trials, the limbs slowed down toward the target, but in unsuccessful cases they moved with almost the same speed at all times. It seems that the successful neural circuits learned to move slowly when fine control is needed. Also, another interesting behavior was observed. When approaching the target after picking up the tool, the limbs first aligned the AC_{end_eff} angle (φ_1) to the AC_{target} angle (φ_2), and then reduced the difference between AC_{end_eff} distance (d_1) and AC_{target} distance (d_2), by extending the limb toward the target.

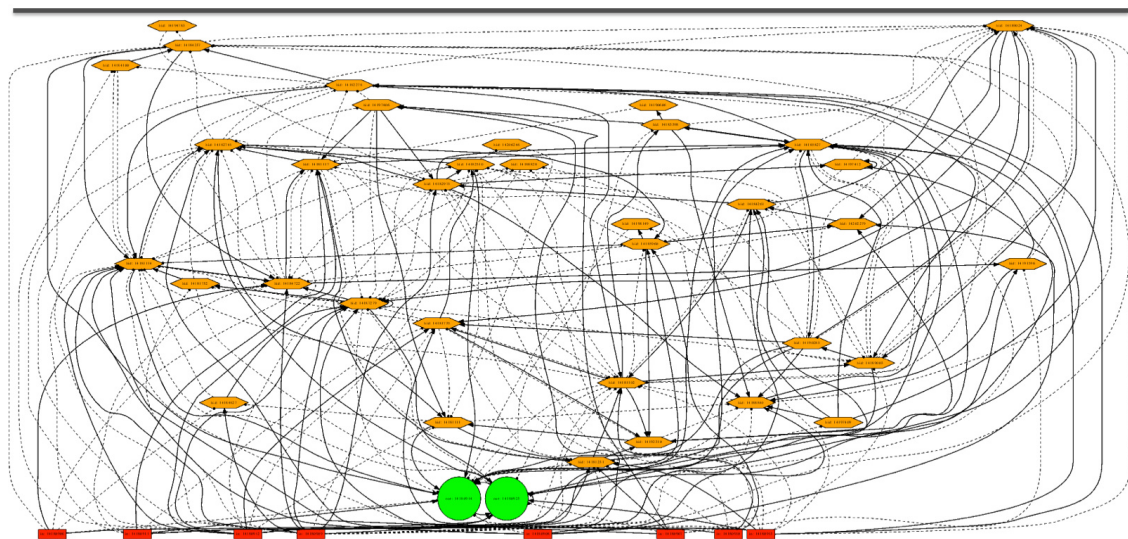
4.4.2 Success Rate for Each Fitness Criterion

When measuring the performance of the different fitness criteria, comparing the raw fitness scores is not fair because different fitness criteria produce different fitness score scales. Therefore, we used success rate to compare the performance between different criteria. Success rates are the percentage of successful target reach events during the 1,000 test trials.

Sensory Input Neurons:  Hidden neurons:  Motor outputs: 



(a) Evolved neural circuit with fitness= S^2T .



(b) Evolved neural circuit with fitness= DS .

Figure 4.6: Examples of evolved neural circuits. (a) Evolved neural circuit with fitness criterion S^2T (speed squared and tool pick-up frequency): 24 neurons (8 sensory inputs, 2 motor outputs, and 14 hidden neurons) and 90 connections (49 excitatory and 41 inhibitory connections; 10 are recurrent). (b) Evolved neural circuit with fitness criterion DS (distance and speed): 45 neurons (8 sensory inputs, 2 motor outputs, and 35 hidden neurons) and 252 connections (124 excitatory and 128 inhibitory connections; 10 are recurrent).

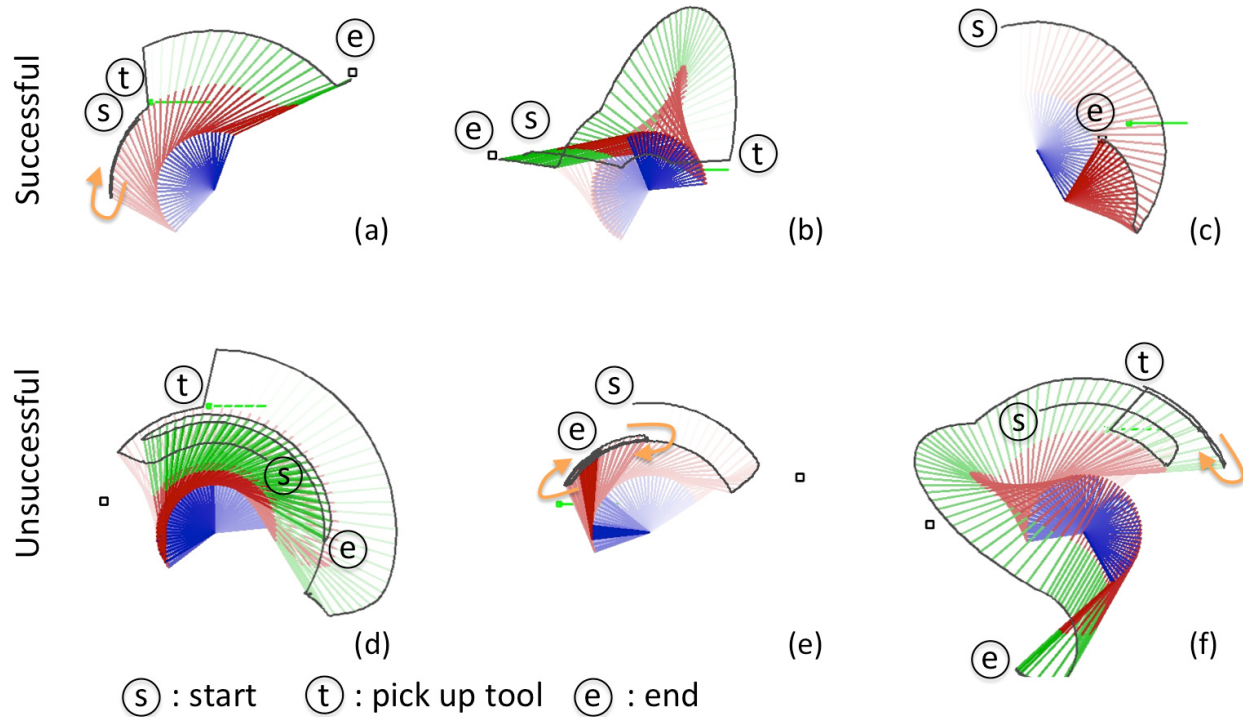


Figure 4.7: Time-lapse images of representative limb movements. The three images in the top row ((a), (b), and (c)) show examples of successful movements (with fitness criteria S^2T). In the movements of (a) and (b), the limbs (blue and red lines) first move towards the tool (green horizontal line) to pick it up (i.e., the limb is extended), and then move towards the target (\square). In the movement of (c), the limb moves directly toward the target without picking up the tool. The trajectories of the end effectors are shown as black curves, and parts of the trajectories that may be hard to keep track of are annotated with orange \rightarrow . The three images in the bottom row ((d), (e), and (f)) show examples of unsuccessful movements (using fitness criteria S^2T without recurrent connections), where the limbs failed to reach the target.

For each fitness criterion, we selected the best neural circuit from the last generation and then ran 1,000 trials. The entire process of evolution and testing was repeated four times ($n = 4$). The results are summarized in Fig. 4.8.

In general, DT , ST , and DST showed superior success rates than their counterparts that did not use the tool pick-up frequency criterion T (D , S , and DS). Within the same group (within $\{DT, ST, DST\}$ or within $\{D, S, DS\}$), the success rates were similar (t-test, $n = 4$, $p > 0.1$ in all cases). However, differences across the two groups were statistically significant (t-test, $n = 4$, $p < 0.01$ for all cases).

The fitness criteria only using D (distance) or S (number of steps) did not show good performance (around 50%), neither did the combination DS . However, combining with T (tool pick-up frequency) boosted the performance (DT, ST , around 70%). This indicates that giving reward for simply picking up the tool, even without any further implications given, could lead to the emergence of adaptive tool-use. Related observation was reported in an experimental study. Amant and Wood [5] noted that in the experiment in Visalberghi and Limongelli [133], capuchin monkeys frequently achieved tasks requiring tool-using ability, by quickly trying many inappropriate strategies with tools (e.g., pick up something and wield it without a clear plan).

In most cases, the fitness criterion S (number of steps) seems to be useful (since high S fitness also means target has been reached, early). The limbs evolved with S (such as ST and DST) reached the target directly without picking up the tool, if the distance to the target is within the limb's reach. For the fitness criteria without S but with T (such as DT), we observed that the limb mostly tried to get to the tool first even when the target is within reach.

As mentioned above, about 50% of the trials were within-reach condition and the other 50% out-of-reach. However, note that even in cases where the success rates are around 50%, the tool was used to reach out-of-reach targets. That means performance of about 50%

included cases where tool was used successfully to reach out-of-reach targets, and cases where within-reach targets were not reached. For example, the success rates for the beyond-reach trials were D (17.78%), S (31.66%), and DS (28.65%) for the fitness conditions lacking T; and DT (93.70%), ST(90.47%), DST(94.07%) for the conditions with T. In sum, D/S/DS conditions did utilize the tool, although less frequently than DT/ST/DST.

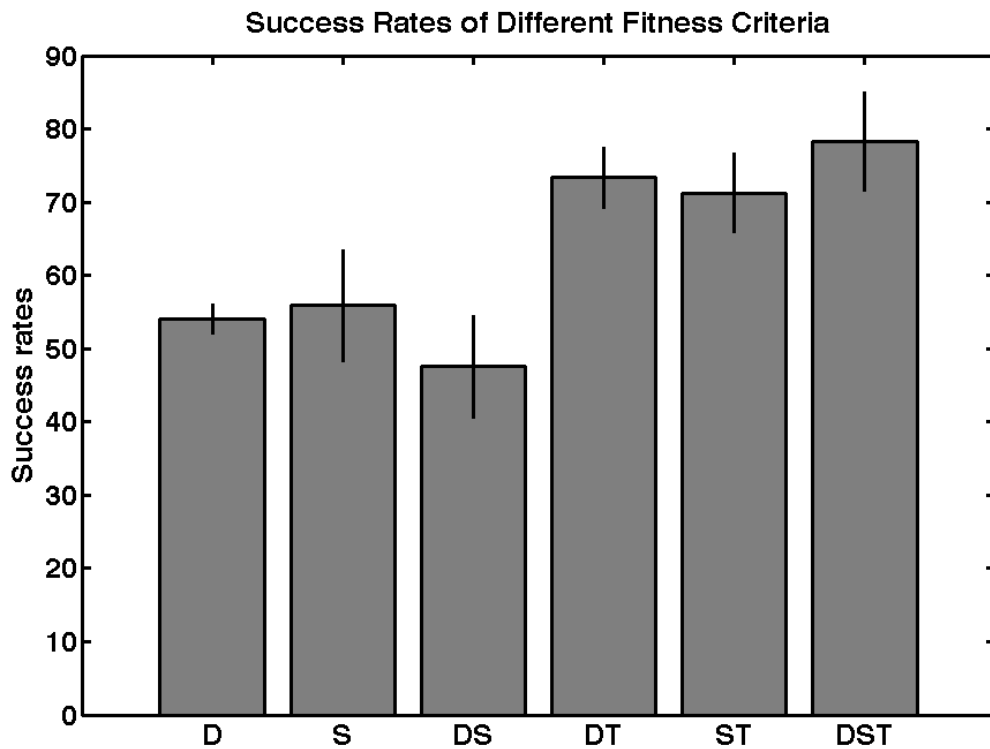


Figure 4.8: Average success rates of controllers based on different fitness criteria. Four sets of experiments (evolution of the neural circuit controller followed by testing) were ran for each fitness criterion. In general, fitness criteria that included T (tool pick-up frequency) did significantly better than those that did not include T (t-test, $n = 4$, $p < 0.01$). Results with S^2T were similar (data not reported here). Note that the target was placed within the limb's reach only $\sim 50\%$ of the time during the trials. See text for details.

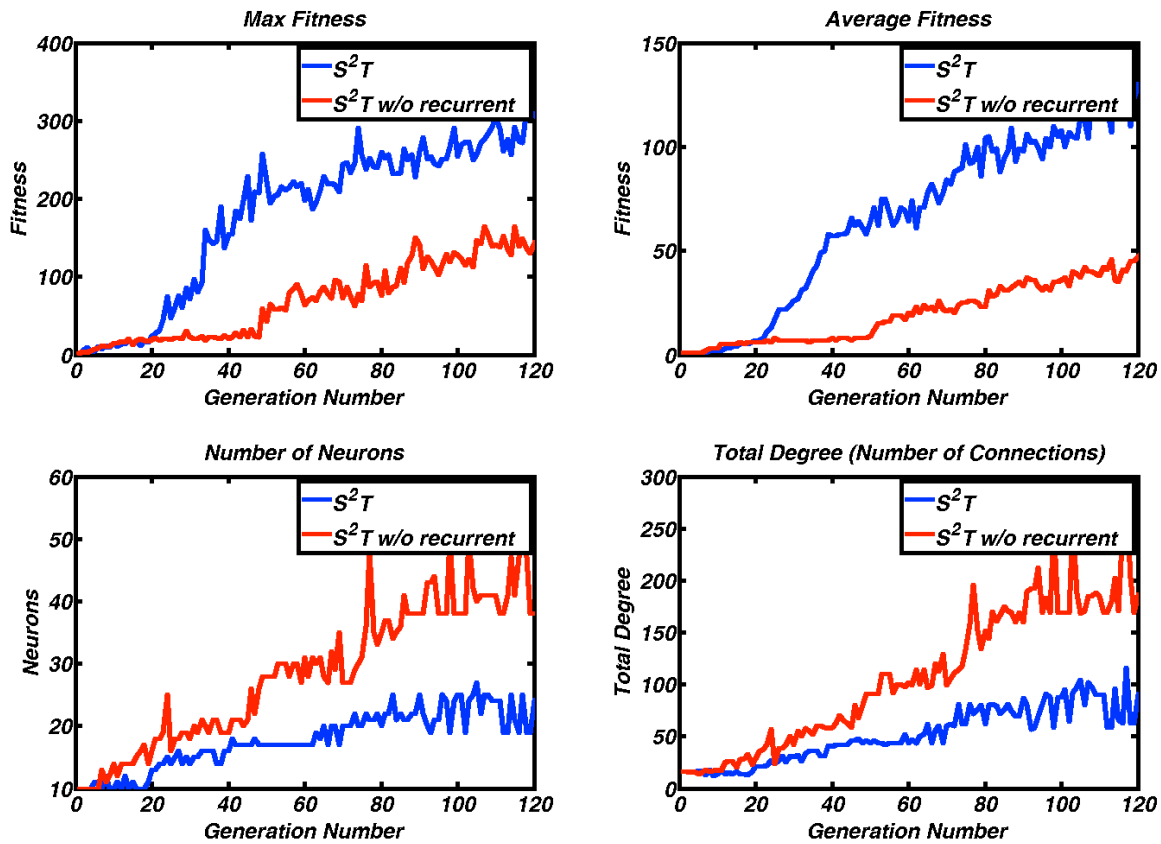


Figure 4.9: Fitness over generations and network size, with or without recurrent connections. S^2T with (blue line) and without recurrent connections (red line) are compared. Top-left: max fitness scores through the generations. Top-right: average fitness scores. Bottom-left: number of total neurons. Bottom-right: number of total degrees (number of connections) in the neural circuits through the generations.

4.4.3 Contribution of Recurrent Connections

We compared neural circuits evolved with and without recurrent connections. The S^2T fitness criterion was used to evolve the neural circuit. In the first case recurrent connections were allowed (S^2T), and in the second case only feedforward connections were allowed ($S^2T_{noRecur}$). First, we compared the success rates of the best chromosomes for the two conditions (S^2T : mean = 80.68%; $S^2T_{noRecur}$: mean = 52.58%). The difference was statistically significant (t-test, n=4, $p < 0.01$). Next, we compared the maximum fitness scores, average fitness scores, the number of total neurons, and the total degrees (the total number of connections) throughout the generations. In Fig. 4.9, top-left and top-right panels show the maximum fitness scores and the average fitness scores throughout the generations when evolving the neural circuits. The one with recurrent connections shows better performances throughout. However, interestingly, the number of neurons and connections in the neural circuits shows the opposite trend, as shown in the bottom-left and the bottom-right panel. The neural circuit without recurrent connections has more neurons and more connections even though the controller (neural circuit) shows significantly lower performance than the one with recurrent connections. This observation emphasizes the importance of recurrent connections in evolving neural circuit controllers with continuous action and feedback loop. Recurrent connections in neural networks can provide memory of the past. Also, recurrent connections can provide the ability to predict future internal dynamics, which is an important ability for neural circuit controllers, especially for challenging control tasks [69].

Also, we analyzed the correlation between (1) the number of recurrent connections and (2) success rates under different fitness conditions (Fig. 4.10). Self loop, 2-hop, and 3-hop loops were counted as recurrent connections. The six fitness criteria compared earlier were analyzed (D , S , DS , DT , ST , and DST). For each fitness criterion, the best neural circuits from each generation (total of 120 per condition, generation 1 to generation 120) were analyzed to

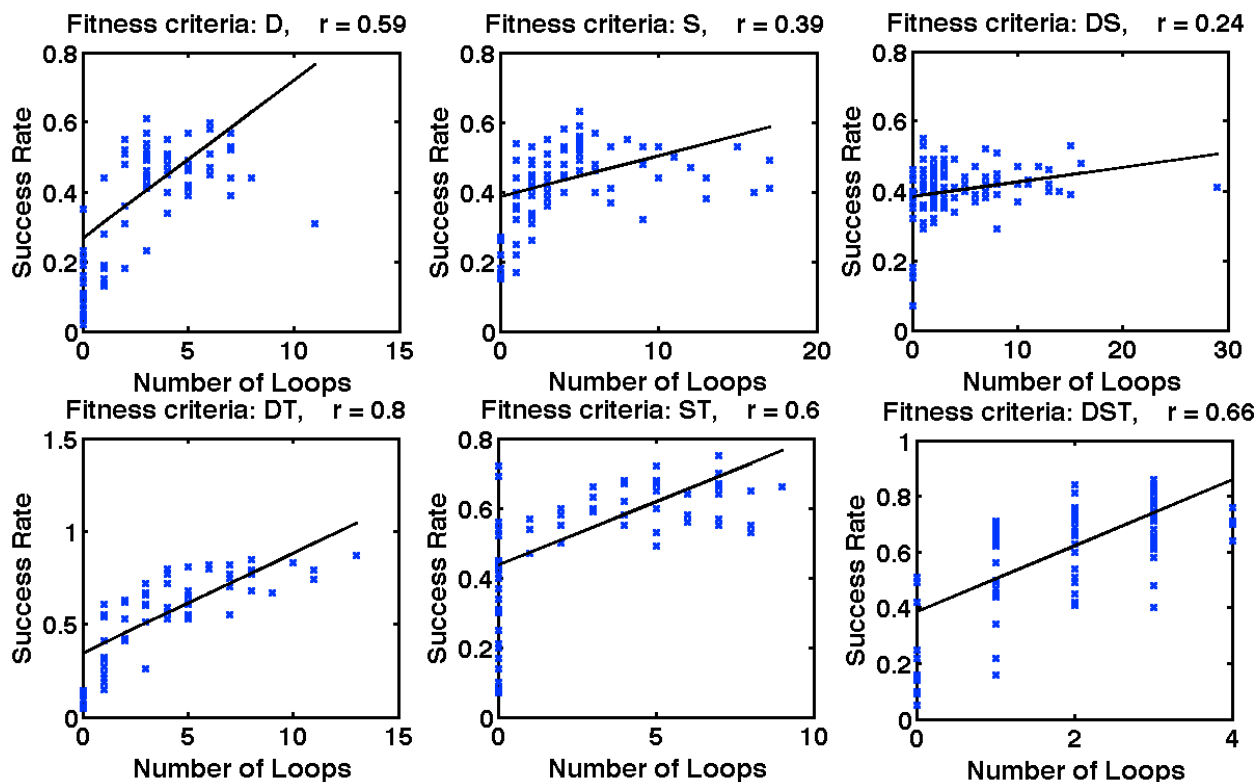


Figure 4.10: Correlations between the number of recurrent connections (self loop + 2 hops + 3 hops) and success rates. The data points (blue “x”), the linear regression lines, and correlation coefficients (r) are plotted, for the six fitness criteria. For each fitness criteria, the 120 best neural circuits for each generation (generation 1 to 120) were analyzed to see the correlation between the number of recurrent connections and success rates (average of 1,000 trials each). The total number of cycles (number of loops) is found to be positively correlated with success rate. Note that the correlation is even higher for those that included T (tool pick-up frequency) in the fitness.

see the correlation between the number of recurrent connections and success rates (same as before). The correlation plots show a trend that the fitness criteria with comparably high correlation coefficients (such as DT , ST , and DST , the ones with T [tool pick-up frequency]) have high success rates in Fig. 4.8. This indicates that the recurrent connections affect positively the performance of the evolved neural circuits (or vice versa). However, the number of recurrent connections themselves may not be critical for the performance as long as recurrence is allowed. Instead, how they are connected could be more important: recurrent connections that are able to predict future internal dynamics [69] could result in neural circuits with higher success rates.

4.4.4 Internal Dynamics

Studying evolutionary autonomous agents (EAAs) has important potentials to understand structure, function, and behavior of biological neural systems [107]. Analysis of neural information processing including internal neurons dynamics in EAAs can provide insights on the function of biological nervous systems. However, even with EAA, fully understanding each neuron’s role and their connection to behavior is not a trivial task, especially as the network size increases. In this section, we present some preliminary analysis of the internal dynamics of evolved neural circuits and correlate the dynamics with behavior.

Fig. 4.11 shows time series values of the fifteen hidden neurons of an evolved neural circuit. In the time series correlation matrix (Fig. 4.11(b)), we can observe four groups of neurons that are highly correlated. The four clusters are cluster 1 (hid14415511, hid14416983, hid14420563), cluster 2 (hid14417371, hid14422164), cluster 3 (hid14415634, hid14415631), and cluster 4 (hid14418199, hid14422660). As we can see in Fig. 4.11(c) and (d), the groups of neurons respond (change activity) to the behavioral events ①, ②, ③, and ④. At the event ①, the limb changes the movement towards the tool, and at ③, it changes towards the target. Tool pickup happens at the event ②, and the limb reaches the target at ④.

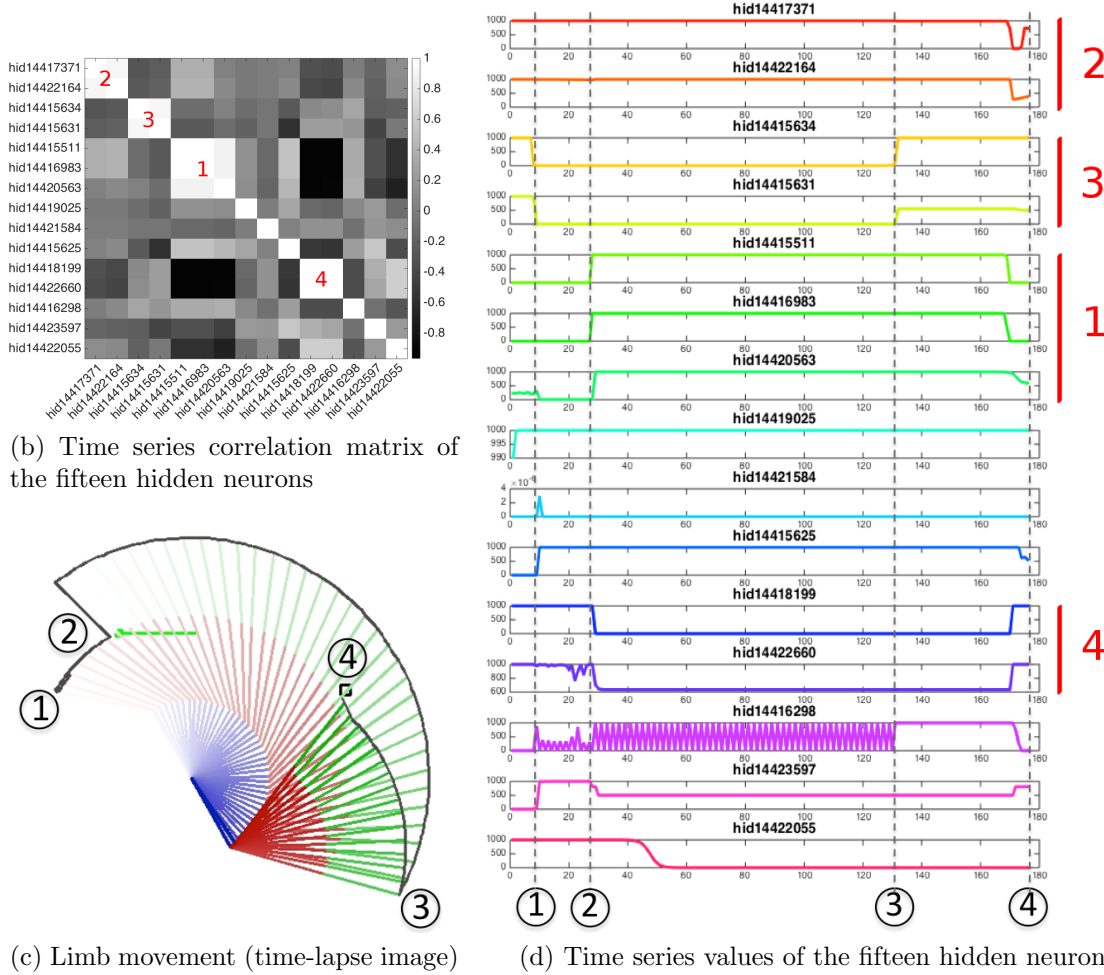
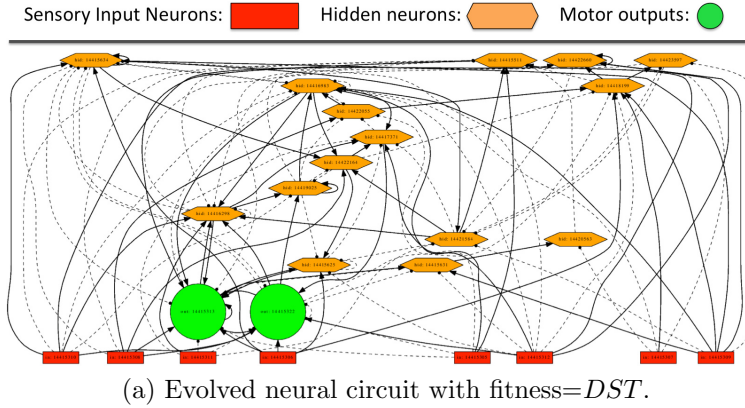


Figure 4.11: Internal dynamics of fifteen hidden neurons, their correlations, and matching limb movements. (a) Evolved neural circuit with fitness = DST . (b) Time series correlation matrix of the fifteen hidden neurons. (c) Time-lapse image of the limb behavior. (d) Time series values of the hidden neurons. Event in (c) are marked with dashed lines. See text for details.

Cluster 3 responds to ① and ③, while cluster 4 to ② and ④. There are single neurons that also exhibit interesting behavior, e.g., hid14416298 (third row from the bottom of Fig. 4.11d showing rapid oscillation between events ② and ③).

4.5 Discussion

The main contribution of this chapter is two-fold: (1) we showed that tool use behavior can be evolved with minimal, indirect fitness criteria, and (2) we found correlations between recurrent connections and tool-use behavior. In most prior tool-use research, the tool was either attached to the agent by default or hard-coded representations were used for the tool. In our case, the distinction between tool and target object was blurred by design, and whether the tool (or the target object) should be reached first was not dictated at all. Despite this paucity of information, our neural controller was able to successfully perform the target reaching task. Furthermore, we found that recurrent connections are key to the success (perhaps since memory of events is needed, e.g., tool picked up, then can reach the target).

The evolving neural circuit controllers also can be utilized for connectomics study. Studying the connectome - a complete neural diagram of the brain - is a challenging problem due to the complexity and scale (see [24] for a review). Synthetic connectomics can benefit natural connectomics research by developing analysis methods based on behavior analysis, internal dynamics analysis, lesion studies, and social context analysis [26]. As our preliminary analysis showed (Section IV. C and D), the synthetically evolving neural circuit controllers can be a good resource for developing such analysis methods for connectomics.

4.6 Conclusion

In this chapter, we investigated how the capability to use a simple yet non-trivial tool can spontaneously emerge in an evolved neural controller for a two degree-of-freedom articulated limb in a target-reaching task. For evolution of the controllers, we used a neural circuit

evolution algorithm that permits incrementally changing topology (NEAT). Our results show that minimal, indirect fitness criteria such as distance to goal (D), speed to reach the goal (S), and tool pick-up frequency (T) are enough to give rise to tool-use behavior. Although the controller was not aware or not made aware of the significance of tool fetching event, inclusion of T in the fitness significantly improved performance. We also found that among the successful neural circuit controllers, those with more recurrent loops were even more effective. When recurrent connections were prohibited, performance suffered. We expect our results to help us better understand the origin of tool-use and the kind of neural circuits that enabled such a powerful trait.

5. END-TO-END TOOL USE LEARNING IN PHYSICS SIMULATION WITH DEEP REINFORCEMENT LEARNING

5.1 Introduction

In this chapter, a more complex tool-use environment is implemented in a physics simulator. It is demonstrated that complex tool-use can be successfully learned with stepwise composite reward shaping using a deep reinforcement learning method. In addition to the background and motivation about the tool-use in AI and robotics (Sec. 4.1.2), this chapter focuses on a more specific area, deep reinforcement learning, applied to tool-use.

In recent years, deep reinforcement learning (RL) have demonstrated good performance in challenging domains such as Atari games [83], Go [115], and continuous control problems [37]. However, the tasks that consists of many subtasks in specific order are still challenging for deep RL. This is because in reinforcement learning an agent first should have some chance to visit the predefined goal states by random exploration to get some rewards, otherwise it will never get a chance to update its value or policy functions. Therefore, for tasks requiring many subtasks to be completed in specific order, there is a very low chance of getting appropriate rewards, if there is no guidance. For example, discrete action environments such as arcade video games [83, 14], *Super Mario Bros*, *VizDoom*, *Montezuma's Revenge*, and *Minecraft* are identified to have such challenges [66, 94, 59]. These problems cannot be solved just using the baseline deep RL methods.

Especially for domains with continuous action spaces, the use of RL is not well explored. Algorithm development and their benchmarks are conducted mostly on tasks without sequential subtask completion requirements. Cutting-edge deep RL algorithms for continuous action spaces are typically developed and benchmarked on physics simulation tasks such as *Swimmer*, *Hopper*, *Walker*, *Half-Cheetah*, *Ant*, *Simple/Full humanoid locomotions*, *Reacher*,

and *Inverted Pendulum* [20, 37, 54], in addition to the simple control tasks such as cart-pole balancing and mountain-cart tasks. The physics-based control tasks above are about learning multidimensional control actions at a given time step to get higher rewards, but none of the tasks consist of sequential subtasks to be completed in a specific order.

Given the background above, this dissertation makes two contributions. First, a tool-use environment is proposed that consists of four subtasks to be completed in a specific order. The goal of the task is to drag a distant object to a target location using a T-shaped tool by controlling an articulated arm (three joints) with a gripper (one joint). The task is motivated by the animal experiment in [78] (Fig. 4.1). To complete the task successfully, the gripper needs to reach the tool handle, grab the tool, move the tool’s end-effector to the object, and then finally move the object to the target location using the tool. The task is implemented in MuJoCo physics simulator [130] for continuous action space control with OpenAI Gym toolkit [20]. This tool-use task environment could be easily used for further deep RL algorithm development and benchmarks. Environments involving tools are nearly absent in the deep RL literature, especially for continuous action spaces.

Second, stepwise composite reward shaping is investigated for learning the multiple subtasks in a specific order in the tool-use task. It is demonstrated that the tool-use task can be successfully learned using stepwise composite reward shaping with a cutting edge deep RL algorithm called Actor Critic using Kronecker-Factored Trust Region (ACKTR). The stepwise composite reward schemes are compared to two other reward schemes to show the effect of stepwise composite reward shaping. Also, the experiments show that the difficulty of the tool-use task can be easily adjusted by selecting one of the four reward schemes, which also could be utilized for further deep RL algorithm development and benchmarks.

5.2 Approach / Method

In this section, first the tool-use environment implemented in a MuJoCo physics simulator [130] is described (Sec. 5.2.1). Then, the various reward elements and four different reward schemes using the elements including the stepwise composite reward schemes are described (Sec. 5.2.2). Finally, the deep RL method, actor-critic policy gradient method using Kronecker-factored trusted region (ACKTR), and the neural network architecture used in this experiment are described (Sec. 5.2.3).

5.2.1 Tool-use Environment in a Physics Simulator

I implemented the tool-use task environment in MuJoCo physics simulator [130] for continuous action spaces control with OpenAI Gym toolkit [20]. MuJoCo (**M**ulti-**J**oint dynamics with **C**ontact) is a physics simulator for robotics, biomechanics, graphics, etc. for fast and accurate simulation.

As in Fig. 5.2, the tool-use task model consists of an articulated three joints arm (blue) with a gripper (gray) that has one joint, an object (green ball), and a T-shaped tool (pink). The three arm joints are controlled by the torques for the joints which is between 1.0 to -1.0. The gripper joint works like a discrete value where 1.0 is applied to close the gripper if the action input for the joint is greater than 0.0, otherwise -1.0 is applied to open the gripper. All the joints including the object and the tool moves in the xy plane.

The task arena is surround by the four boundaries (top, bottom, left and right) and the goal of the task is to move the object to the bottom boundary using the tool. The tool location is fixed, and the arm is located either on the left or the right side of the tool, and the object locations are randomly chosen out of the arm’s reach for each episode.

To complete the task, four subtasks should be completed in sequential order; (1) the gripper needs to reach the tool handle while the gripper is open; (2) grasp the tool handle; (3) move the tool’s end-effector to the object while keeping the gripper closed to manipulate

the tool; (4) move the object to the target location (bottom boundary of the arena) by maneuvering the tool.

The maximum time step for each episode is set to 500 steps where 1 time step corresponds to 0.01 seconds. Once the final task condition (i.e. moving the object to the target location using the tool) is satisfied, the episode ends early and the remaining steps are calculated as an extra reward for the task completion speed.

5.2.2 Stepwise Composite Reward Shaping

When rewards are sparse it is more challenging to learn the task since the agent needs to rely on random exploration until it accidentally hits a state where rewards can be observed. If the agent has lower chances of getting a reward signal, it also has lower chance of learning. Especially when a task consists of subtasks where a completion of one task is required to complete next subtasks in large action spaces, reaching a final state (where the final task is successfully completed) through naive random exploration is very unlikely to succeed. The tool-use task in this chapter is such a task: The task environment with continuous action spaces consists of four subtasks (reach, grasp, tool-to-object, and object-to-target) to be completed in that specific order.

Therefore, one sparse reward that can only be given when the final task is completed (i.e. move the object to the target) is very unlikely to be observed through random exploration. In addition, shaping rewards for the sequential order subtasks is not trivial, because one reward element for a subtask can be turned into noise and interruption for other tasks.

In this chapter, I propose a stepwise composite reward shaping method following [96] for the tool-use task where the composite rewards for the subtasks are shaped in a stepwise manner. In the following section, the reward elements used for reward shaping are described (Sec. 5.2.2.1 and 5.2.2.2). Next, four different reward schemes used in the experiments are described (Sec. 5.2.2.3): (1) stepwise composite reward with intermediate elements; (2)

stepwise composite reward without intermediate elements; (3) one sparse reward; and (4) all reward elements without stepwise order.

5.2.2.1 Subtask completion reward elements

First, the primitive elements that are used in the reward elements for composite reward shaping are defined below.

- $p^{\vec{G}}$: the pinch position of the gripper.
- $p^{\vec{L}}$: the position of the left claw of the gripper
- $p^{\vec{R}}$: the position of the right claw of the gripper
- $p^{\vec{H}}$: the position of the tool handle.
- $p^{\vec{E}}$: the position of the tool end-effector.
- $p^{\vec{O}}$: the position of the object.
- $p^{\vec{T}}$: the position of the target (the bottom boundary of the arena).

The four subtasks (reach, grasp, tool-to-object, and object-to-target) are considered to be completed if the following conditions are satisfied.

- **Reach-Tool**: The pinch position of the gripper is close enough to the tool handle.

$$ReachTool = \|p^{\vec{G}} - p^{\vec{H}}\|_2 < k_g, \quad (5.1)$$

where k_g is a constant that is set to the half that of the length of the gripper's claws.

- **Grasp-Tool**: The gripper grabs the tool handle.

$$GraspTool = ReachTool \wedge \left(\theta_c < \|p^{\vec{L}} - p^{\vec{R}}\|_2 < \theta_o \right), \quad (5.2)$$

where θ_c and θ_o are thresholds for detecting close and open status of the gripper.

- **Tool-Reach-Object:** The tool end-effector reaches the object.

$$ToolReachObject = GraspTool \wedge \left(\|p^{\vec{E}} - p^{\vec{O}}\|_2 < k_o \right), \quad (5.3)$$

where k_o is a constant that is set to the half that of the length of the tool tip.

- **Object-Reach-Target:** The object reaches the target.

$$ObjectReachTarget = GraspTool \wedge \left(\|p^{\vec{O}} - p^{\vec{T}}\|_2 < k_a \right), \quad (5.4)$$

where k_a is a constant that is set to the half that of the thickness of the arena boundary bar.

5.2.2.2 Intermediate reward elements

The intermediate reward elements are negatively correlated to the distance between the objects in the physics simulation to guide the gripper to the tool, the tool to the object, and the object to the target. The value of these rewards are between 0 and 1 and are defined as follows. These reward elements could guide the learning in a more specific manner, between the completion conditions for the subtasks. (Sec. 5.2.2.1).

- **Distance between Gripper and Tool:** This continuous reward is negatively correlated with the distance between the gripper pinch position and the tool handle position.

$$r_{c1} = 1 - \tanh^2 \left(\frac{\|p^{\vec{G}} - p^{\vec{H}}\|_2}{k_w} \right), \quad (5.5)$$

where k_w is a constant that is set to the width of the arena.

- **Distance between Tool and Object:** This continuous reward is negatively correlated with the distance between the tool end-effector position and the object position.

$$r_{c2} = 1 - \tanh^2 \left(\frac{\|p^{\vec{E}} - p^{\vec{O}}\|_2}{k_w} \right), \quad (5.6)$$

where k_w is the same as above.

- **Distance between Object and Target:** This continuous reward is negatively correlated with the distance between the object position and the target position.

$$r_{c3} = 1 - \tanh^2 \left(\frac{\|p^{\vec{O}} - p^{\vec{T}}\|_2}{k_w} \right), \quad (5.7)$$

where k_w is the same as above.

5.2.2.3 Reward shaping

Using the reward elements described above, the four different reward schemes are defined. The first is the *stepwise composite reward with intermediate elements*, which includes both subtask completion and intermediate reward elements. The second is the *stepwise composite reward without intermediate elements* that only use the subtask completion reward elements. For the first and second reward schemes, the amount of reward between the subtasks do not overlap, where the amount of reward is increased stepwise when one subtask is completed. For example, the maximum reward for the first subtask completion does not exceed the minimum reward for the second subtask. The third one is *one sparse reward*, where the reward is given only when the final task (moving the object to the target using the tool) is completed. Finally, the last one is *all reward elements without stepwise order* where all the reward elements are always provided, but not in a stepwise manner as in the first and the second schemes. Details of each reward shaping scheme are described below.

(1) Stepwise composite reward with intermediate elements

(stepwise-intermediate): This reward scheme consists of subtask completion reward elements (Sec. 5.2.2.1) and the intermediate reward elements (Sec. 5.2.2.2). In this condition, the agent can continuously receive the intermediate reward signals to guide itself to complete the next sequential subtask. The amounts of reward between the sequential subtasks are designed not to be overlap, so that the maximum reward for a subtask does not exceed the minimum reward for the next subtask. The reward is given at each time step and an episode ends when the task is successfully completed (i.e. *ObjectReachTarget* is True). Once the task is successfully completed, a bonus reward 300 is given along with a reward for the speed at which the task is completed.

$$r_{\{p^G, p^L, p^R, p^H, p^E, p^O, p^T\}} = \begin{cases} 300.0 + k_s \times (s_{max} - s_k) & \text{if } ObjectReachTarget \text{ (Eq. 5.4)} \\ 1.5 + 1.5 \times r_{c3} & \text{if } \neg ObjectReachTarget \\ & \wedge ToolReachObject \text{ (Eq. 5.3)} \\ 0.25 + 1.25 \times r_{c2} & \text{if } \neg ObjectReachTarget \\ & \wedge \neg ToolReachObject \\ & \wedge GraspTool \text{ (Eq. 5.2)} \\ 0.125 & \text{if } \neg ObjectReachTarget \\ & \wedge \neg ToolReachObject \\ & \wedge \neg GraspTool \\ & \wedge ReachTool \text{ (Eq. 5.1)} \\ 0 + 0.125 \times r_{c1} & \text{otherwise} \end{cases} \quad (5.8)$$

where s_{max} is the maximum time step for each episode (500 steps) and s_k is the time steps so far. k_s is a constant for weighting the speed of the task completion, which is set to 3.0. r_{c1} , r_{c2} , and r_{c3} are the intermediate reward elements that are negatively correlated with the distance between the gripper and the tool (Eq. 5.5), between the tool and the object (Eq. 5.6), and between the object and the target (Eq. 5.7), respectively.

(2) **Stepwise composite reward without intermediate elements (stepwise-no-**

intermediate): This reward scheme only consists of the subtask completion reward elements (Sec. 5.2.2.1) such that the intermediate continuous reward elements are not used. In this reward scheme, the agent only gets increased reward when it completes the next sequential subtask. The other conditions are the same as in (1).

$$r_{\{p^{\vec{G}}, p^{\vec{L}}, p^{\vec{R}}, p^{\vec{H}}, p^{\vec{E}}, p^{\vec{O}}, p^{\vec{T}}\}} = \begin{cases} 300.0 + k_s \times (s_{max} - s_k) & \text{if } ObjectReachTarget \text{ (Eq. 5.4)} \\ 3.0 & \text{if } \neg ObjectReachTarget \\ & \wedge ToolReachObject \text{ (Eq. 5.3)} \\ 1.5 & \text{if } \neg ObjectReachTarget \\ & \wedge \neg ToolReachObject \\ & \wedge GraspTool \text{ (Eq. 5.2)} \\ 0.125 & \text{if } \neg ObjectReachTarget \\ & \wedge \neg ToolReachObject \\ & \wedge \neg GraspTool \\ & \wedge ReachTool \text{ (Eq. 5.1)} \\ 0 & \text{otherwise} \end{cases} \quad (5.9)$$

where s_{max} , s_k , and k_s are defined and initialized as above.

- (3) **One sparse reward (one-sparse)**: The reward is given only when the final subtask is completed, where the agent moves the object to the target location using the tool. Otherwise, the agent receives zero reward. Once the task is completed, it will receive the reward 300 plus the reward for the speed at which the task is completed, just like the others.

$$r_{\{p^{\vec{G}}, p^{\vec{L}}, p^{\vec{R}}, p^{\vec{H}}, p^{\vec{E}}, p^{\vec{O}}, p^{\vec{T}}\}} = \begin{cases} 300.0 + k_s \times (s_{max} - s_k) & \text{if } ObjectReachTarget \\ 0 & \text{otherwise} \end{cases} \quad (5.10)$$

where s_{max} , s_k , and k_s are defined and initialized as above.

- (4) **All reward elements without stepwise order (no-stepwise-all)**: In this reward scheme, all reward elements are always provided, but not in a stepwise manner. Until completing the final task, the agent receives the combination of the all reward elements except the bonus reward for the final task. Once the final task is completed, the bonus reward of 300 along with the task completion speed reward is given.

$$r_{\{p^{\vec{G}}, p^{\vec{L}}, p^{\vec{R}}, p^{\vec{H}}, p^{\vec{E}}, p^{\vec{D}}, p^{\vec{T}}\}} = \begin{cases} 300.0 + k_s \times (s_{max} - s_k) & \text{if } ObjectReachTarget \\ 0.125 \times r_{c1} + 1.25 \times r_{c2} + 1.5 \times r_{c3} & \text{otherwise} \end{cases} \quad (5.11)$$

where s_{max} , s_k , k_s , r_{c1} , r_{c2} , and r_{c3} are defined and initialized as above.

5.2.3 Deep RL Method: Actor-Critic Policy Gradient Method using Kronecker-Factored Trust Region

For training the model, a deep reinforcement learning method called Actor Critic using Kronecker-Factored Trust Region (ACKTR) [149] was used. ACKTR is a policy gradient method using actor critic architecture with natural gradient, which was recently proposed as a more efficient (in sample complexity) policy gradient reinforcement learning method than other cutting-edge policy gradient methods such as Asynchronous Advantage Actor Critic (A3C) [82] or Trust Region Policy Optimization (TRPO) [112].

The following sections describe details about the ACKTR method including a brief description about policy gradient methods, actor critic methods, natural gradient, and applying the method for continuous action spaces. Lastly, the neural network architecture used in this dissertation and the overview of the method are described. Some parts of the descriptions about policy gradient methods and actor critic methods follow [124, 125].

5.2.3.1 Policy Gradient Methods

In reinforcement learning, *policy gradient methods* directly learn a parameterized policy that produces actions without considering values of the actions, while *action value methods*

such as Deep Q-Network (DQN) [83] and its variants [53, 136, 132] learn action values that are used to select actions.

There are some advantages of policy gradient methods over action value methods as described in [125]. Policy gradient methods can learn probabilities of taking actions, learn the proper amount of exploration, and approach asymptotically to deterministic policies, which is not possible with the ϵ -greedy and the action value methods. Policy gradient methods more generally work on various tasks than action value methods, while action value methods are more sample efficient when they work. Also, as mentioned in [113], DQN has shown good performance with discrete action spaces such as Atari games [83, 14], but has not shown good performance with continuous control spaces such as in [20, 37].

Policy gradient methods learn the policy parameters according to the gradient of some performance measure $J(\boldsymbol{\theta}_t)$ with respect to the policy parameters. The methods update the gradient ascent approximation to maximize the performance. In general, a method following Eq. 5.12 is called policy gradient method.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}, \quad (5.12)$$

where $\boldsymbol{\theta}_t$ is the policy parameter vector at time t . $\nabla J(\boldsymbol{\theta}_t)$ is the stochastic estimate that approximates the gradient of the performance measure $J(\boldsymbol{\theta}_t)$, with respect to $\boldsymbol{\theta}_t$, and $\widehat{}$ represents the expected value. α is a step size for the update.

To approximate $\nabla J(\boldsymbol{\theta}_t)$ in Eq. 5.12, the *policy gradient theorem* (Eq. 5.13) establishes an analytical expression for the gradient of the performance measure with respect to the policy parameters, without computing the derivative of the state distribution.

$$\nabla J(\boldsymbol{\theta}_t) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}), \quad (5.13)$$

where π is the policy corresponding to parameter vector $\boldsymbol{\theta}$, and the gradients are vectors of

```

1: Initialize the policy parameter  $\theta$ 
2: for counter = 0 to maximum iteration  $M$  do
3:   Sample an episode  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ , following  $\pi$ 
4:   for each step  $t = 0, \dots, T - 1$  do
5:      $G \leftarrow$  return from step  $t$ 
6:      $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_{\theta} \ln \pi(A_t | S_t, \theta)$ 

```

Figure 5.1: REINFORCE algorithm pseudocode [125]

partial derivatives with respect to θ . The policy $\pi(a|s, \theta)$ is the probability of taking action a at time t given a state s at time t with policy parameter θ . $q_{\pi}(s, a)$ is the value of taking action a in state s under policy π . The term μ is the on-policy distribution following π . The notation \propto means ‘proportional to’.

From Eq. 5.12 and 5.13, the REINFORCE algorithm [141] leads to Eq. 5.14.

$$\theta_{t+1} \equiv \theta_t + \alpha G_t \frac{\nabla_{\theta} \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}, \quad (5.14)$$

where at time t , A_t is the sampled action by $\pi(a|S_t, \theta)$ in the state S_t given the policy parameter θ . G_t is the return (cumulative discounted reward) following time t . The symbol \equiv means ‘is equal by definition to’. The REINFORCE algorithm uses full return from time t , including all future rewards until the end of the episode. In other words, the REINFORCE algorithm is a Monte Carlo algorithm that performs all updates while looking back at the values stored at each time step after the episode is completed. The pseudocode of the REINFORCE algorithm is shown in Fig. 5.1, where $\nabla_{\theta} \ln \pi(A_t | S_t, \theta)$ is the compact form of $\frac{\nabla_{\theta} \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}$ in Eq. 5.14, using the identity $\nabla \ln x = \frac{\nabla x}{x}$.

5.2.3.2 Actor-Critic Method

Modifying the REINFORCE algorithm to incorporate a baseline (the return G_t is subtracted by the baseline $b(S_t)$ [141, 124], as follows) reduces the variance of the estimates so that the learning speed increases.

$$\boldsymbol{\theta}_{t+1} \equiv \boldsymbol{\theta}_t + \alpha (G_t - b(S_t)) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}, \quad (5.15)$$

To estimate a baseline, a state value function $\hat{v}(S_t, \mathbf{w})$ can be learned, where \mathbf{w} is a weight vector for the value function. In an actor-critic method, a state value function is used as a baseline and also for bootstrapping. Bootstrapping means that a state value function is used for estimating the subsequent state's values. By modifying Eq. 5.15, a k-step actor-critic method can be formulated as follows [82, 125].

$$\boldsymbol{\theta}_{t+1} \equiv \boldsymbol{\theta}_t + \alpha^\theta (G_{t:t+k} - \hat{v}(S_t, \mathbf{w})) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \quad (5.16)$$

$$= \boldsymbol{\theta}_t + \alpha^\theta A^\pi(S_t, A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}, \quad (5.17)$$

where α^θ is a step size for the policy parameter vector $\boldsymbol{\theta}$ update. $A^\pi(S_t, A_t)$ is called the *advantage* function. The advantage function for a k-step method can be generalized as,

$$A^\pi(S_t, A_t) = \sum_{i=0}^{k-1} \gamma^i R(S_{t+i}, A_{t+i}) + \gamma^k \hat{v}(S_{t+k}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \quad (5.18)$$

where $\hat{v}(S_t, \mathbf{w})$ is 0 if S_t is the terminal state of the episode.

The value function parameter vector \mathbf{w} can be updated using the advantage estimates

as:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha^W A^\pi(S_t, A_t) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}), \quad (5.19)$$

where α^W is a step size for the \mathbf{w} update.

5.2.3.3 Actor Critic with Natural Gradient using K-FAC

Although policy gradient methods with deep neural networks have demonstrated good performance in control tasks such as Atari games and continuous action control tasks, there is a challenge of choosing the right step size for updating the parameters. If a step size is too small, the learning is seriously slow. If the step size is too large, it can be overwhelmed by the noise of the function approximation and/or cause a catastrophic degradation in performance [106]. Choosing an inappropriate step size adversely affects reinforcement learning more than supervised learning, because a bad step size can cause a bad policy which in turn collects bad samples under the bad policy from the next batch. As a consequence, it cannot recover from the bad policy and bad sampling cycle, so that performance collapses.

To attack this problem, instead of using stochastic gradient descent (SGD) that is commonly used in deep NN and deep RL, trust region policy optimization (TRPO) [112] was proposed, where the surrogate objective function is maximized, subject to a constraint or a penalty for step sizes for policy update, as follows.

- With a constraint:

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}}(a_t | s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t | s_t)} \hat{A}_t^\pi \right] \quad (5.20)$$

$$\text{subject to} \quad \hat{\mathbb{E}} [\text{KL}[\pi_{\boldsymbol{\theta}_{old}}(\cdot | s_t), \pi_{\boldsymbol{\theta}}(\cdot | s_t)]] \leq \delta, \quad (5.21)$$

- With a penalty:

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t|s_t)} \hat{A}_t^\pi - \beta \text{KL}[\pi_{\boldsymbol{\theta}_{old}}(\cdot|s_t), \pi_{\boldsymbol{\theta}}(\cdot|s_t)] \right], \quad (5.22)$$

where $\boldsymbol{\theta}_{old}$ is the policy parameter vector before the update, \hat{A}_t^π is the advantage function estimator at time step t , and KL is the Kullback-Leibler (KL) divergence [67] between the two probability distributions to measure how one diverges from the other. $\pi_{\boldsymbol{\theta}}(\cdot|s_t)$ is the policy distribution of all actions given a state s at time step t with the policy parameter $\boldsymbol{\theta}$. In continuous action space, $\pi_{\boldsymbol{\theta}}(\cdot|s_t)$ is a multidimensional normal distribution as described in Sec. 5.2.3.4; in discrete action space, it is a multinomial distribution over the discrete actions. Linear approximation to the objective (Eq. 5.20) and the quadratic approximation to the KL term are used, which is called the natural policy gradient. However, TRPO, in practice, shows poor performance for large neural network models suffering sample inefficiency because the approximations above for the conjugate gradient requires large number of samples for each batch update and large amount of computation.

Alternatively, a recently proposed Kronecker-factored approximated curvature (K-FAC) can be used, which is a scalable approximation to the Fisher information matrix, a local quadratic approximation to the KL divergence, to perform efficient approximation for the natural gradient updates [79, 48]. This technique showed it can speed up training of large neural network models in supervised learning, where a running average of curvature information is kept to enable to use of small samples for the batch update. Motivated by this work, a scalable trust region optimization algorithm using K-FAC for actor-critic methods called Actor Critic using Kronecker-Factored Trust Region (ACKTR) was proposed [149]. In this dissertation, the ACKTR algorithm is used for updating both the policy network (actor) and the value network (critic).

5.2.3.4 Policy Parameterization for Continuous Actions

Policy gradient methods provide a useful way to tackle continuous action spaces where infinite number of actions exist. Selecting actions for discrete action spaces and for continuous action spaces should be different. For discrete action spaces, numerical preferences for each action in a state are the outputs of the policy network. From the numerical preferences, the actions with highest probabilities in each state are selected according to an exponential softmax distribution. However, for continuous action spaces, instead of using learned probabilities for each action, the outputs of the policy network are two real number vectors; one for the mean μ and the other for the standard deviation σ of a multidimensional normal distribution as used in [72, 125]. The vector size (the number of normal distributions) matches the number of actions. In the tool-use environment that consists of three arm joints and one gripper joint, the number of normal distributions is four. The probability density function for the normal distribution is:

$$f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad (5.23)$$

where π in this equation is the mathematical constant (≈ 3.141592). The actions are sampled from the multidimensional normal distributions following μ and σ vectors from the policy network. The last layer of the policy network for the μ vectors can be a linear layer, while the approximation for σ can be an exponential of the linear output since the standard deviation should be a positive real number.

5.2.3.5 Neural Network Architecture

For the policy network architecture, three dense hidden layers are used where the size of 64, 64, and 32 neurons are used for the hidden layers h1, h2, and h3, respectively. For the hidden layers, a tanh activation function is used. The output dense layer consists of

8 neurons, and no activation function is applied (linear units). Exponential functions are applied to the four of the output values to represent the standard deviation σ .

For the value network architecture, three dense hidden layers are used as well, where the size of 64, 64, and 32 neurons are used for the hidden layers h1, h2, and h3, respectively. The exponential linear unit (ELU) [30] activation function is applied for the hidden layers. The output is one neuron densely connected to the hidden layer h3, without an activation function.

Finally, Fig. 5.2 shows the overview diagram of the actor-critic architecture in the task. At time t , the reward r_t and state S_t are sampled from the tool-use environment. The observed state S_t are fed into the policy network (the actor with parameter vector $\boldsymbol{\theta}$) and the value network (the critic with parameter vector \mathbf{w}). Next, the policy network produces the stochastic policy $\pi(A_t|S_t, \boldsymbol{\theta})$ that consists of the mean μ and the standard deviation σ vectors of the four dimensional normal distributions. The actions A_t for each joint (the motor force vector for the joints) are sampled from the multidimensional normal distributions. The sampled action vector A_t are fed into the advantage function and used to take the next action step as well. Also, the value network estimates the value of the state $\hat{v}(S_t, \mathbf{w})$ which is fed into the advantage function. The advantage function $A^\pi(S_t, A_t)$ keeps the values of A_t , $\hat{v}(S_t, \mathbf{w})$, and the reward r_t for k time steps, then use them to update the policy and value parameters ($\boldsymbol{\theta}$ and \mathbf{w}) using the natural gradient for the trust region update. The environment receives the sampled action vector A_t , takes the next step, and produces the new state and reward (S_{t+1} and r_{t+1}).

5.3 Experiments

The model was trained with each of the four reward schemes. For each of the four reward schemes, five model instances were trained simultaneously with five different random seeds for about 24,000 iterations (about 2 days and several hours) on a machine equipped with

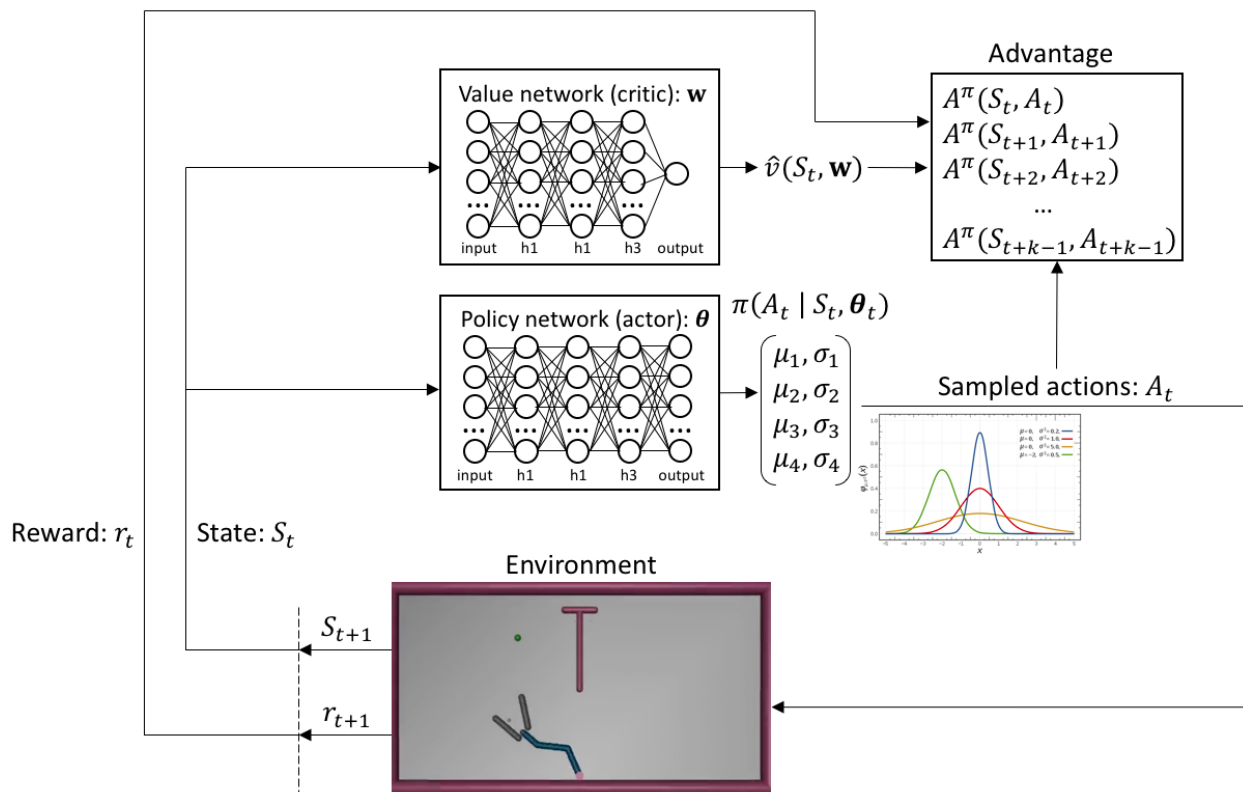


Figure 5.2: Overview of the actor-critic architecture with the tool-use environment. At time t , reward r_t and state S_t are sampled from the tool-use environment. The observed state S_t are fed into the policy network (the actor with parameter vector θ) and the value network (the critic with parameter vector \mathbf{w}). Next, the policy network produces the stochastic policy $\pi(A_t|S_t, \theta)$ that consists of the mean μ and the standard deviation σ vectors of four dimensional normal distributions. The actions A_t for each joint (the motor torque vector for the joints) are sampled from the multidimensional normal distributions. The sampled action vector A_t are fed into the advantage function and used to take the next action. Also, the value network estimates the value of the state $\hat{v}(S_t, \mathbf{w})$ which is fed into the advantage function. The advantage function $A^\pi(S_t, A_t)$ keeps the values of A_t , $\hat{v}(S_t, \mathbf{w})$, and the reward r_t for k time steps, then use them to update the policy and value parameters (θ and \mathbf{w}) using the natural gradient for the trust region update. The environment receives the sampled action vector A_t , take a next step, and produces the new state and reward (S_{t+1} and r_{t+1}). See text in Sec. 5.2.3 for details.

an Intel Core i7 CPU (6700K Quad Core) and an NVIDIA TITAN X GPU (Pascal). One iteration consisted of 2,500 steps so that the total steps taken for the training were about 60 million. For the implementation of the experiments, OpenAI Gym [20], baselines [34], and TensorFlow [1] were utilized.

After training the models, the best models with the highest episode reward means were selected for each of the four reward schemes. Next, the selected models were compared by running them with the trained policy networks for 1,000 episodes. Success rates, episode rewards, and episode lengths were compared. In addition, those performance measures were compared for all five trained models as well.

The number of maximum time steps for an episode was set to 500 steps, where 1 time step simulates 0.01 seconds. If the task completion condition (i.e. moving the object to the target location using the tool) is completed before reaching the maximum time steps, the episode ends early and the remaining time steps are counted as an extra reward for the task completion speed. During the training of each model, the episode reward means and the episode length means were recorded.

The input vector (the observed state S_t at time t) for the model consisted of the positions and velocities of the four joints, $p^{\vec{G}}$, $p^{\vec{H}}$, $p^{\vec{E}}$, $p^{\vec{O}}$, $\|p^{\vec{L}} - p^{\vec{R}}\|$, $\|p^{\vec{G}} - p^{\vec{H}}\|$, $\|p^{\vec{E}} - p^{\vec{O}}\|$, and $\|p^{\vec{O}} - p^{\vec{T}}\|$. The input vectors were normalized using running estimates of the means and standard deviations. In each time step, there was a relatively small penalty in reward that is proportional to the forces applied to the joints.

5.4 Results and Analysis

This section presents the results of training the models with the four different reward schemes. In brief, the *stepwise-intermediate* and *stepwise-no-intermediate* reward schemes could lead to successful learning, with both showing comparable performance. Both *one-sparse* and *no-stepwise-all* reward schemes could not make progress on tool-use learning,

where none of the five instances for each scheme could learn even the first subtask (i.e. Reach-Tool). In the following sections, I will present the behavior analysis and performance comparison of the trained agents, followed by the learning curve comparison.

5.4.1 Behavior analysis of the trained agents

In this section, I will describe qualitative behavior analysis of the trained models. The *stepwise-intermediate* and *stepwise-no-intermediate* reward schemes could train agents that can complete the tool-use task successfully, while none of the agents trained with the other *one-sparse* and *no-stepwise-all* schemes could complete the task successfully even for the first subtask (i.e. Reach-Tool).

Fig. 5.3 shows an example of the tool-use task episode from the learned model with *stepwise-intermediate* (the model marked with the blue learning curve in Fig. 5.9), where the screen shots of the subtask completions are shown. Fig. 5.3a shows initial locations of the arm, tool, and object. The tool location is the same for all episodes. The object location is selected by random beyond the arm’s reach and the arm can be located either to the right or to the left side of the tool. To complete the first and the second subtasks (i.e. Reach-Tool and Grasp-Tool), the agent first opens the gripper and bends the arm while approaching the tool handle (Fig. 5.3b). When approaching the tool handle, the agent should learn to approach the tool handle in the proper direction so that the gripper does not hit the tool. Moving the gripper directly to the tool without bending the arm from the start positions causes it to hit the tool. As a consequence, the tool moves away to a distance that the arm cannot reach as shown in Fig. 5.4. Therefore, to prevent this, the agent learned to bend the arm first and then approach the tool to place the tool handle between the grippers (Fig. 5.3c). Note that I did not give any specific guidance for this kind of behavior. When the gripper reaches the tool handle, the agent closes the gripper to grasp the tool and then manipulates the tool toward the object while keeping the gripper closed (Fig. 5.3d and 5.3e).

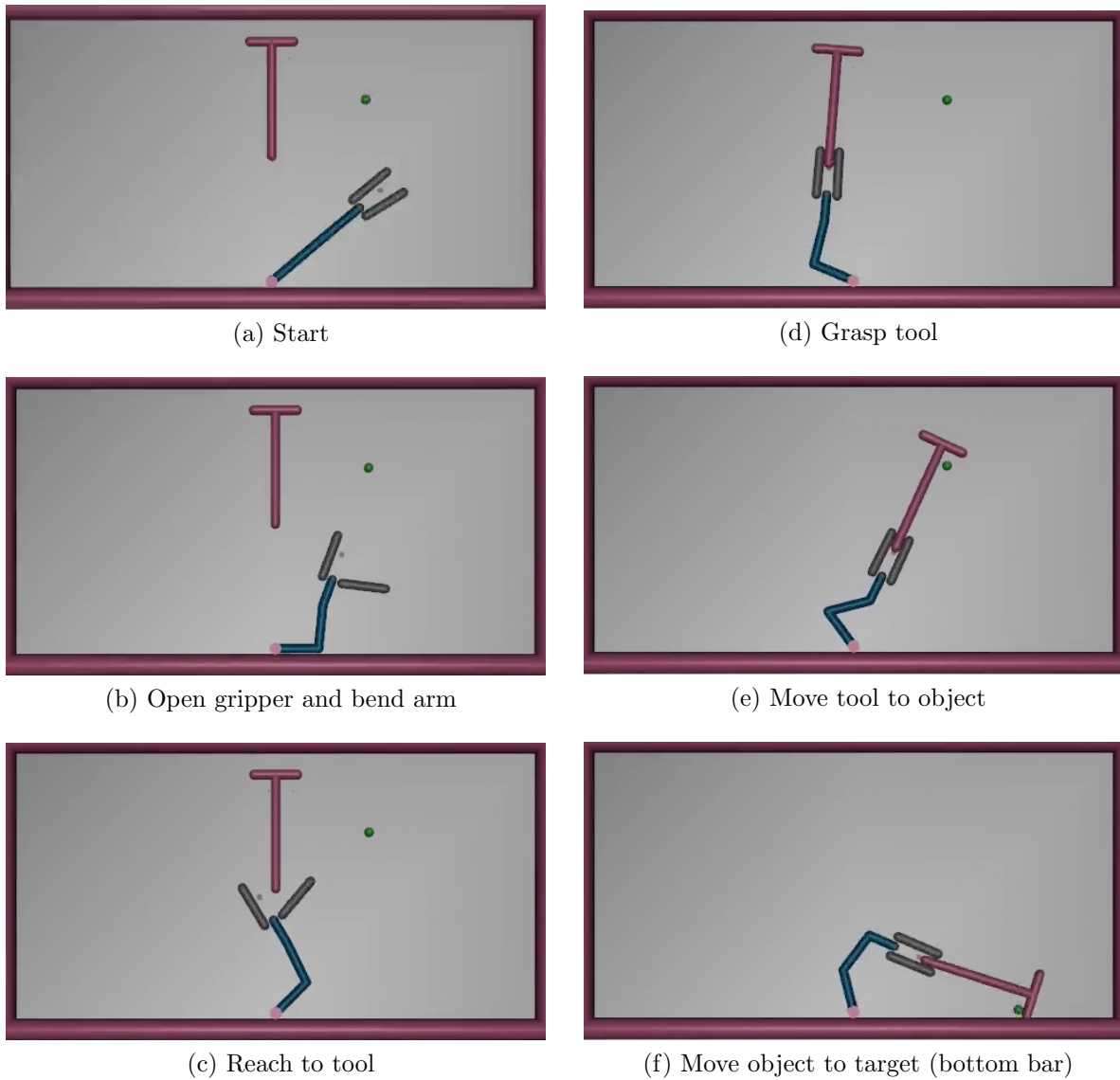


Figure 5.3: Successful example of the tool-use task, where the screen shots of the subtasks completion are shown. (a) shows initial locations of the tool, object and arm. For each episode, the object location is randomly selected out of the arm reach, the tool location is fixed, and the arm location is either right or left side of the tool. (b) Open the gripper and bend the arm. (c) Reach the tool handle. (d) Grasp the tool. After grasping the tool, the agent need to keep the gripper closed to manipulate the tool. (e) Move the tool (tool end-effector) to the object. (f) Move the object to the bottom bar (target) using the tool. See text for details.

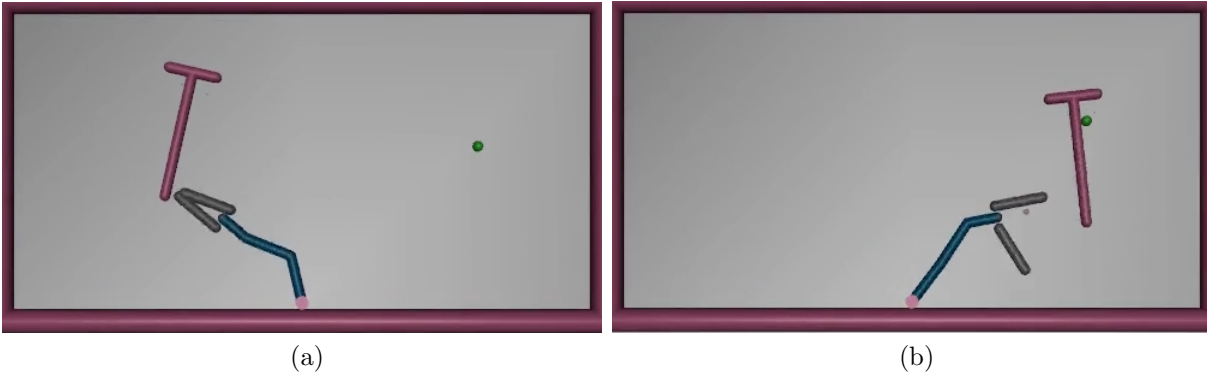


Figure 5.4: Examples of unsuccessful episodes. In both screen shots (a) and (b), the arm hit the tool and as a result the tool moves away to a distance that the arm cannot reach.

Then, finally it moves the object to the target location (the bottom boundary) using the tool (Fig. 5.3f)

The agents trained with *one-sparse* and *no-stepwise-all* could not learn to complete even the first subtask Reach-Tool. The agents trained with *one-sparse* converged to one type of behavior that is to stay in the same location. This behavior is to minimize the reward penalty of the joint forces, which is in turn a locally optimal behavior because no other reward signals are given until the last subtask Object-Reach-Target is completed. Stumbling into the solution for the last subtask by random exploration happens to be very unlikely in tasks such as tool-use where several subtasks need to be satisfied in a specific order. On the other hand, the agent trained with *no-stepwise-all* did not show a convergence to a certain behavior type. They tend to move randomly because of the noisy reward signal that is irrelevant to the completion of the first subtask Reach-Tool.

The agents with successful task learning (trained with *stepwise-intermediate* and *stepwise-no-intermediate*) developed different behavioral strategies. The differences in behavior of a successful agent are not due to different reward schemes but because of the different random seed. Under different random seeds, the initial locations in the parameter space are different

and they ended up in good enough local optima with. For example, one type of successful behavior was that once the agent grasps the tool, the agent directly moves towards the object location. In another type of behavior, the agent extends the arm after grasping the tool then moves to the left boundary and then to the right boundary. While the first type looks like an appropriate way of completing the task, the second type also completes the task. Interestingly, while the agent with the first type learns to complete the tasks faster than the other type (Fig. 5.7), the second type showed a little higher success rate (Fig. 5.5) by developing stable controls with fewer mistakes than the first type. The overall average rewards were a bit higher for the first type than the second type (Fig. 5.6) due to the shorter episode length leading to higher reward.

5.4.2 Comparing performance of the trained agents

In this section, first, the best models for each reward scheme were selected and run with the trained policies for 1,000 episodes. From the test running results, the average of success rates, episode rewards, and episode lengths were compared. In brief, the best models from *stepwise-intermediate* and *stepwise-no-intermediate* reward schemes show much better performance than the other two models from *one-sparse* and *no-stepwise-all*. The first two models showed higher success rates, higher episode rewards, and shorter episode lengths than the latter two models. The differences between the first two models were very subtle. These results indicate that the reward shaping in a stepwise manner is effective for the tasks such as tool-use that consists of several subtasks arranged in specific order. Also, it indicates that the intermediate reward elements (e.g. continuously presenting distances to subtask goal statuses) are not necessarily helpful once stepwise reward shaping is applied appropriately. In addition to the comparison of the best models, all five models trained with different random seeds for each scheme were compared. The results are plotted and shows similar trends to the best model comparison. The details are described in the rest of the

section.

Fig. 5.5 shows the success rate of the best models for each reward scheme. The best models from the five different random seeds for each of the four reward schemes were selected and ran for 1,000 episodes with the trained policy networks. An episode was counted as success if the last subtask (Object-Reach-Target) is satisfied. The bar-plots show the success rates of each reward scheme’s best model. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (success rate: 93%) and *stepwise-no-intermediate* (success rate: 96%) show success rates over 90% where the latter one is slightly higher than the first one. The *one-sparse* and *no-stepwise-all* never completed the task.

Fig. 5.6 shows the mean episode reward of the best models’ 1,000-episode run for each scheme. For this comparison, the reward scheme *no-stepwise-all* was used to make sure all intermediate continuous rewards are always given even though subtasks are not completed. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (mean reward: 1615.4) and *stepwise-no-intermediate* (mean reward: 1607.3) show much higher reward means than the *one-sparse* (mean reward: 794.2) and *no-stepwise-all* (mean reward: 776.3). The *no-stepwise-all* (std: 88.1) has a larger standard deviation than *one-sparse* (std: 50.4) even though they both could not learn the first subtask (Reach-Tool) and have similar mean reward. This is because the agent *one-sparse* learning was converged to minimize joint movement (staying without much moving) while the *no-stepwise-all* learning could not converge to any behavior (kind of random moves).

Fig. 5.7 shows the mean episode length of the best models. An episode ends early when the episode satisfies the last subtask (Object-Reach-Target) before it reaches the maximum time steps (500 steps). The bar-plots show the mean episode length of each scheme’s best model. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-*

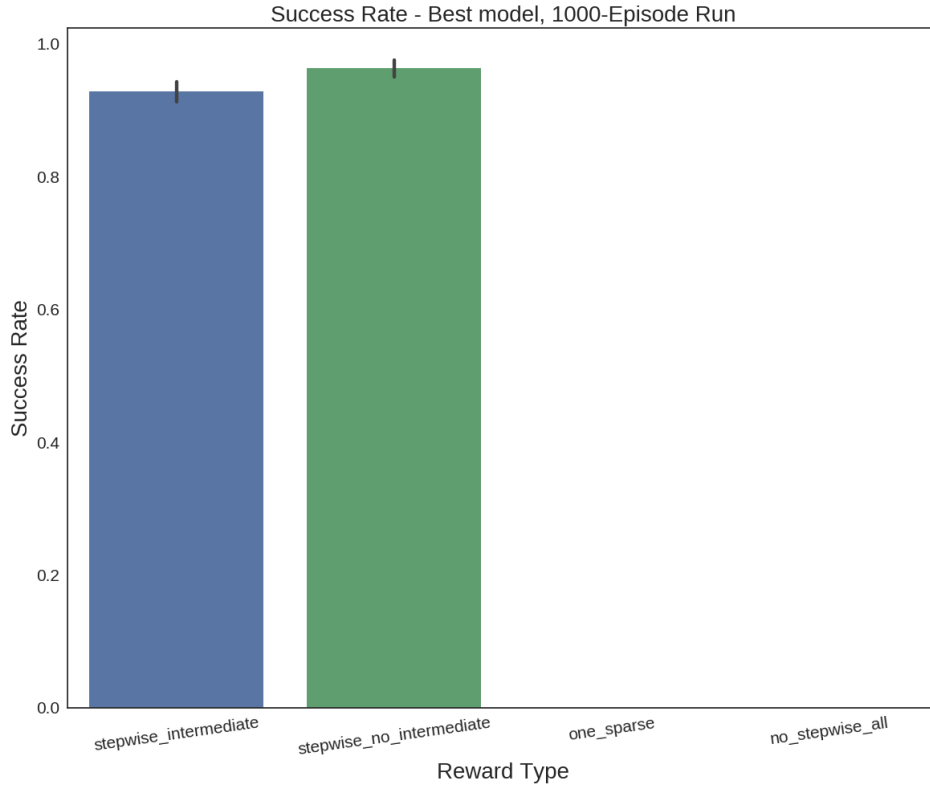


Figure 5.5: Success rate comparison for the best models. The best models from the five different random seeds for each of the four reward schemes were selected and ran for 1,000 episodes with the trained policy networks. An episode was counted as success if the last subtask (Object Reach Target) is satisfied. The bar-plots show the success rates of each reward scheme’s best model. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (success rate: 93%) and *stepwise-no-intermediate* (success rate: 96%) show success rates over 90% where the latter one is slightly higher than the first one. The *one-sparse* and *no-stepwise-all* never completed the task.

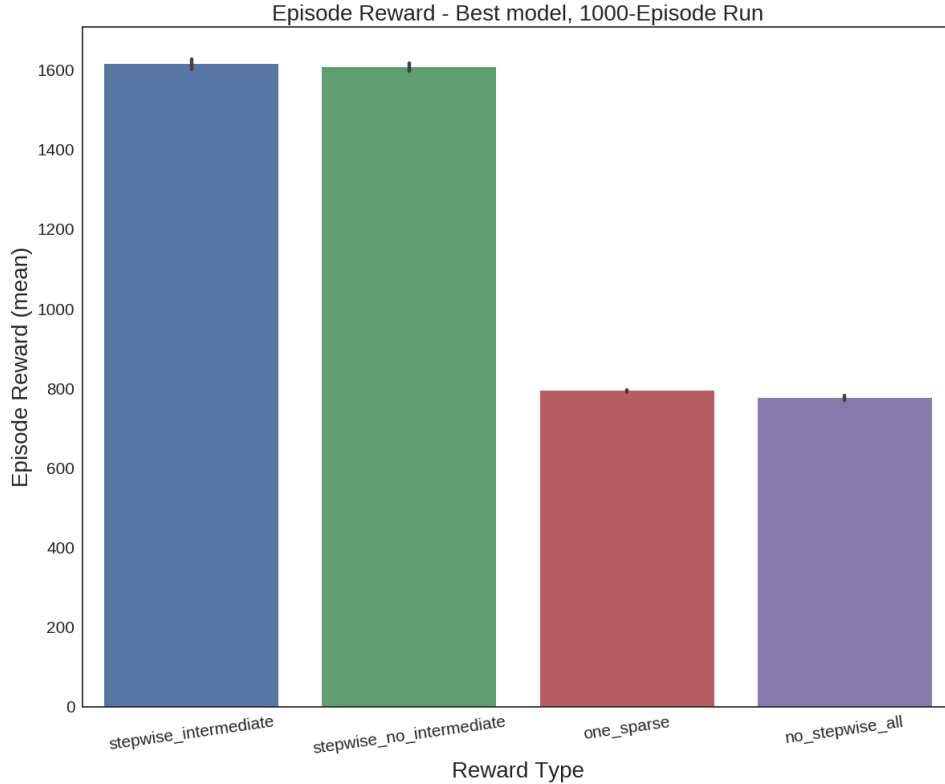


Figure 5.6: Mean episode reward for the best models. The best models for each reward schemes were selected and run for 1,000 episodes with the trained policies. The bar-plots show the mean episode reward of each scheme’s best model. For this comparison, the reward scheme *no-stepwise-all* was used to make sure all kinds of rewards are always given even though subtasks are not completed. From the leftmost bar to the right, the results of the *stepwise-intermediate*, *stepwise-no-intermediate*, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (mean reward: 1615.4) and *stepwise-no-intermediate* (mean reward: 1607.3) show much higher reward means than the *one-sparse* (mean reward: 794.2) and *no-stepwise-all* (mean reward: 776.3). The *no-stepwise-all* has a larger standard deviation than *one-sparse* even though they both could not learn the first subtask (Reach Tool) and have similar mean reward. This is because the agent *one-sparse* learning was converged to minimize joint movement (staying without much moving) while the *no-stepwise-all* learning could not converge to any behavior (kind of random moves).

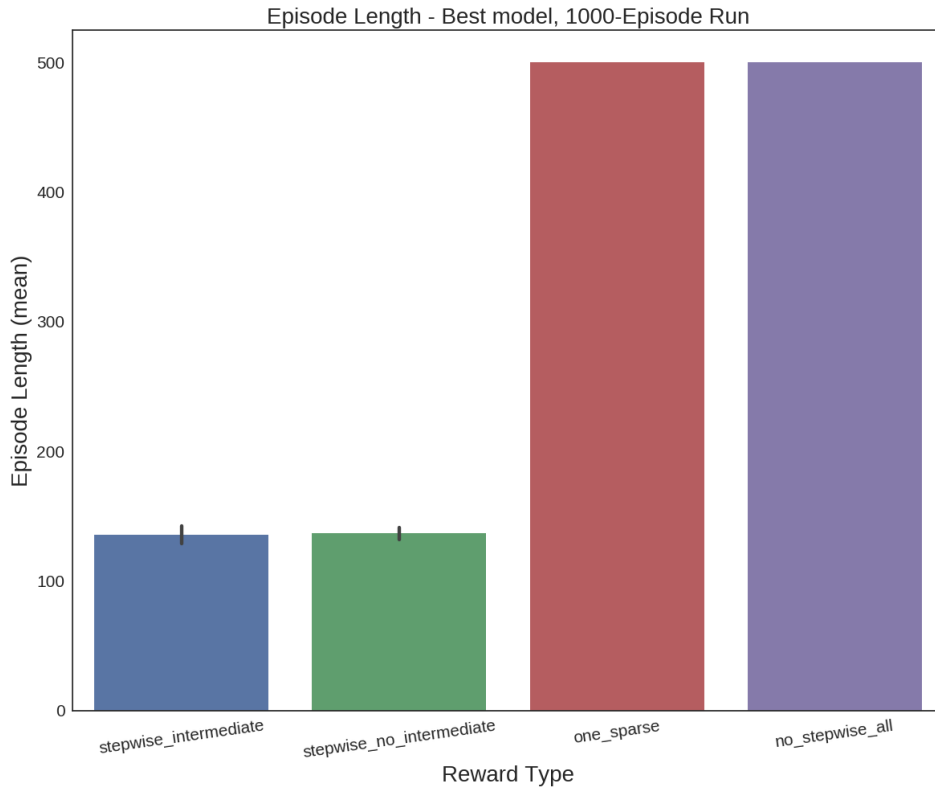
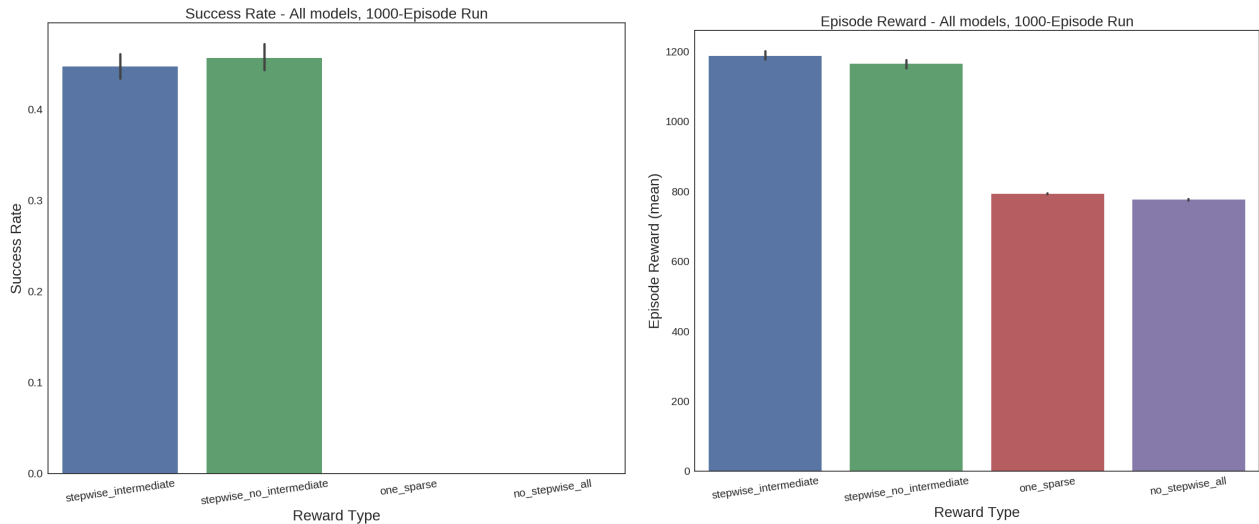
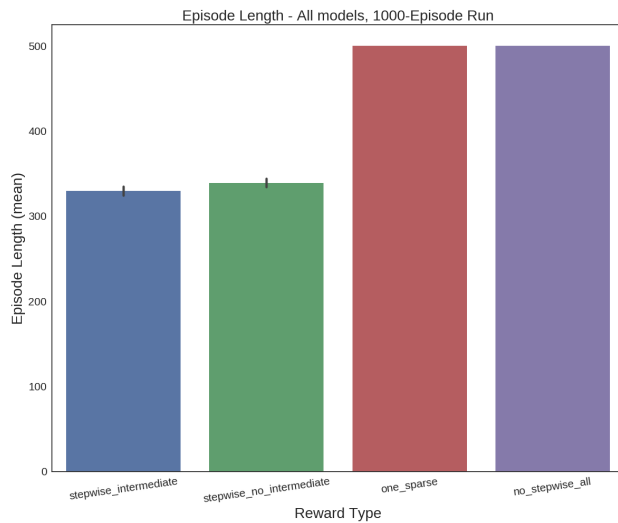


Figure 5.7: Mean episode length for the best models. The best models for each reward scheme were selected and run for 1,000 episodes with the trained policies. An episode length ends early when the episode satisfies the last subtask (Object Reach Target) before it reaches the maximum time steps (500 steps). The *stepwise-intermediate* (mean length: 135.5 steps) and *stepwise-no-intermediate* (mean length: 136.8 steps) show shorter mean episode length than the maximum steps where the first one shows slightly shorter mean episode length. This may be the reason why the *stepwise-intermediate* scheme shows better mean reward than *stepwise-no-intermediate* in Fig. 5.6 even though the one has lower success rate than the latter one in Fig. 5.5. The best model from *stepwise-intermediate* learned a behavior where the tool end-effector is moved directly towards the object while the *stepwise-no-intermediate* learned a behavior where the tool is moved to the left and right boundary with a full stretch of the arm. However, the first one (*stepwise-intermediate*) makes more mistakes than the latter one (*stepwise-no-intermediate*) when grasping the tool and moving the tool end-effector to the object. In sum, the *stepwise-intermediate* scheme learned to complete the task faster than the *stepwise-no-intermediate*, while the latter learned to perform the task more stably than the former. Note that this behavioral difference did not come from the different reward schemes. The different random seeds played a role in the behavioral differences. We can say that both schemes led to good enough local optima. The *one-sparse* (length mean: 500.0 steps) and *no-stepwise-all* (length mean: 500.0 steps) never completed the tasks.



(a) Success rates

(b) Mean episode reward



(c) Mean episode length

Figure 5.8: Comparison of all five models (including all different random seeds) for each reward scheme. All five models for each scheme trained with different random seeds were compared by running for 1,000 episodes with the learned policies. The trends are similar to the comparison of the best models, where the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show higher success rate, higher mean episode reward, and shorter mean episode length than the *one-sparse* and the *no-stepwise-all* schemes. The differences between the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes are very subtle. The *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show degraded values (lower success rates, lower mean episode reward, and longer mean episode length) compared to the best models because the models here include unsuccessful learnings. The *one-sparse* and the *no-stepwise-all* schemes have almost the same values (success rates, mean episode reward, and mean episode length) to the best models.

no-intermediate, *one-sparse*, and *no-stepwise-all* are plotted. The *stepwise-intermediate* (mean length: 135.5 steps) and *stepwise-no-intermediate* (mean length: 136.8 steps) show shorter mean episode length than the maximum steps where the first one shows slightly shorter mean episode length. This may be the reason why the *stepwise-intermediate* scheme shows a bit better mean reward than *stepwise-no-intermediate* in Fig. 5.6 even though the one has lower success rate than the latter one in Fig. 5.5. The best model from *stepwise-intermediate* learned a behavior where the tool end-effector is moved directly towards the object while the *stepwise-no-intermediate* learned a behavior where the tool is moved to the left and right boundary with a full stretch of the arm. However, the first one (*stepwise-intermediate*) makes more mistakes than the latter one (*stepwise-no-intermediate*) when grasping the tool and moving the tool end-effector to the object. In sum, the *stepwise-intermediate* scheme learned to complete the task faster than the *stepwise-no-intermediate*, while the latter learned to perform the task more stably than the former. Note that this behavioral difference did not come from the different reward schemes. The different random seeds played a role in the behavioral differences. We can say that both schemes led to good enough local optima. The *one-sparse* (length mean: 500.0 steps) and *no-stepwise-all* (length mean: 500.0 steps) never completed the tasks.

The comparison of all five models (including all different random seeds) for each scheme is presented in Fig. 5.8. All five models for each reward scheme trained with different random seeds were compared by running for 1,000 episodes with the learned policies. The trends are similar to the comparison of the best models, where the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show higher success rate, higher mean episode reward, and shorter mean episode length than the *one-sparse* and the *no-stepwise-all* schemes. The differences between the *stepwise-intermediate* and the *stepwise-no-intermediate* schemes are very subtle. The *stepwise-intermediate* and the *stepwise-no-intermediate* schemes show degraded values (lower success rates, lower mean episode reward, and longer mean episode

length) compared to the best models because the models here include unsuccessful learnings. The *one-sparse* and the *no-stepwise-all* schemes have almost the same values (success rates, mean episode reward, and mean episode length) to the best models.

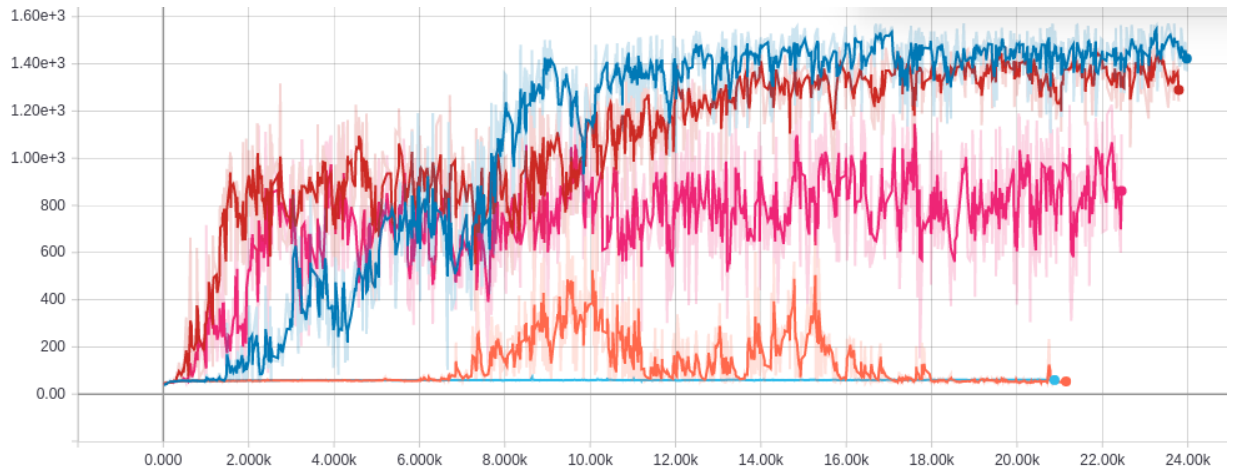
5.4.3 Learning curves

5.4.3.1 Stepwise composite reward with intermediate elements (*stepwise-intermediate*)

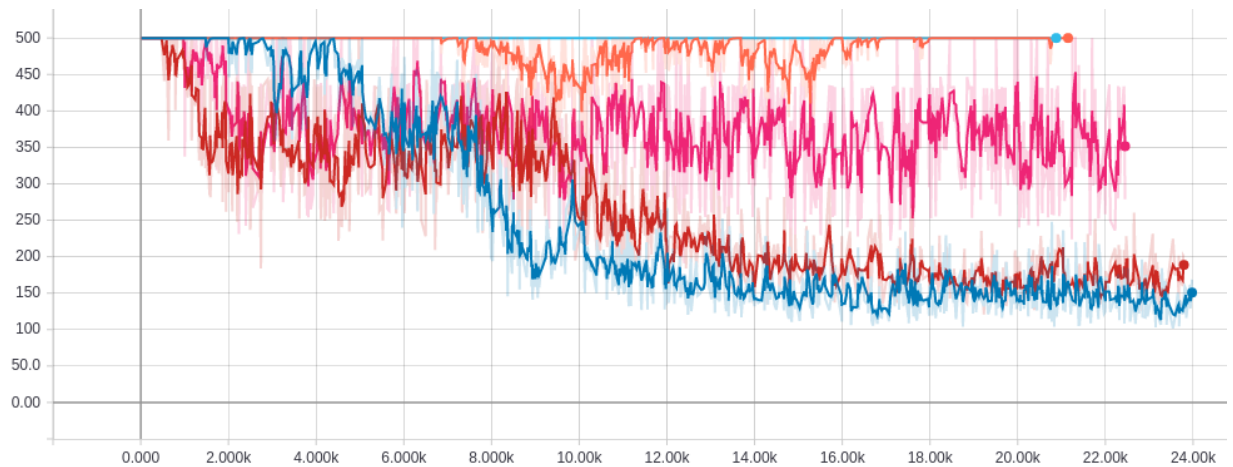
Fig. 5.9 shows the mean episode reward and mean episode length for each iteration during training. The five model instances were trained simultaneously with five different random seeds for 24,000 iterations at most (2 days and 4 hours). Different colors mean different random seeds. One iteration consisted of 2,500 steps so that the total steps taken for the training were 60 millions at most. The reason why different random seeds lead to different total iterations is that some random seeds (e.g. the blue and red learning curves) successfully learned the task so that the successful episodes completed early as in the Fig. 5.9b, while some others (e.g. the orange and sky-blue learning curves) could not learn the task successfully so the episode lengths were equal to the maximum episode steps (500 steps). As stated in [54], deep reinforcement learning is known to show variance in results according to stochasticity in the environments or the learning process (such as random weight initialization), which is an open problem that needs further studies.

Fig. 5.9a shows the mean episode reward during training. The two models (the blue and red learning curves) learned the task successfully after around 12,000 iterations and reached about 1,400 mean reward. The pink case reached an intermediate performance, and then stayed at the plateau. The orange and sky-blue cases could not learn the task successfully. They struggled to learn the reach/grab subtasks and could not make further progress.

Fig. 5.9b shows mean episode length during training. Successful cases (the blue and red learning curves) show shorter episode lengths as they show higher rewards in Fig. 5.9a and converge to a solution about 150 steps, while the others show longer episode length (the pink



(a) Mean episode reward



(b) Mean episode length

Figure 5.9: Learning curve for different random seeds under *stepwise composite reward with intermediate elements (stepwise-intermediate)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 24,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training were 60M at most. The reason why different random seeds lead to different total iterations is that some random seeds (the blue and red learning curves) successfully learned the tasks and complete a task early as in the Fig. 5.9b, while some others (the orange and sky-blue learning curves) could not learn the task successfully so the episode lengths were equal to the maximum episode steps (500 steps). (a) Mean episode reward during training. The two models (the blue and red learning curves) learned the task successfully around 12,000 iterations and reached about 1400 reward. The pink case reached an intermediate performance and stayed at the plateau. The orange and sky-blue cases could not learn the task successfully. (b) Mean episode length during training. Successful cases (the blue and red learning curves) show shorter episode lengths as they show higher rewards, while the others show longer episode length (the pink learning curve) or maximum episode length (500 steps) as they could not reach the target for early episode termination.

learning curve) or maximum episode length (500 steps) as most of the episodes could not reach the target.

5.4.3.2 *Stepwise composite reward without intermediate elements*
(*stepwise-no-intermediate*)

Fig. 5.10 shows the mean episode reward and mean episode length during training with five different random seeds in the reward scheme of *stepwise composite reward without intermediate elements* (Eq. 5.9). Different colors mean different random seeds. The five model instances were trained for 24,000 iterations at most (2 days and 4 hours). One iteration consists of 2,500 steps, so the total steps taken for the training are 60M at most. The pink case successfully learned the task at around 7,000 iterations. The orange case learned gradually and reached successful learning state at around 23,000 iterations. The others could not learn the task. The blue and sky-blue cases could not get out of the plateau, where they could learn the first two subtasks (i.e. Reach-Tool and Grasp-Tool), but could not learn the other two subtasks (i.e. Tool-Reach-Object and Object-Reach-Target). The red case could not show much learning, where it could learn to make the gripper close to the tool handle, but closes the gripper too early so that it was not able to grasp the tool. Fig. 5.10a shows mean episode reward during training. The pink case reach about 1,400 reward score at around 7,000 iterations, and the orange case’s rewards gradually increased and reached about 1,400 at 23,000 iterations. The blue and sky-blue cases reached around 700 and stayed that plateau. The red case’s reward stayed around 60, where the agent could not learn to grasp the tool. Fig. 5.10b shows mean episode length during training. The graph shows inverse correlation with the graph in (a). The pink case learned to complete the task less than 200 steps at around 7,000 iterations, and the orange case gradually speed up the task completion and reach around 200 steps at around 23,000 iterations. The blue and sky-blue cases were stayed around 450 steps, which means some of the episode completed by luck,

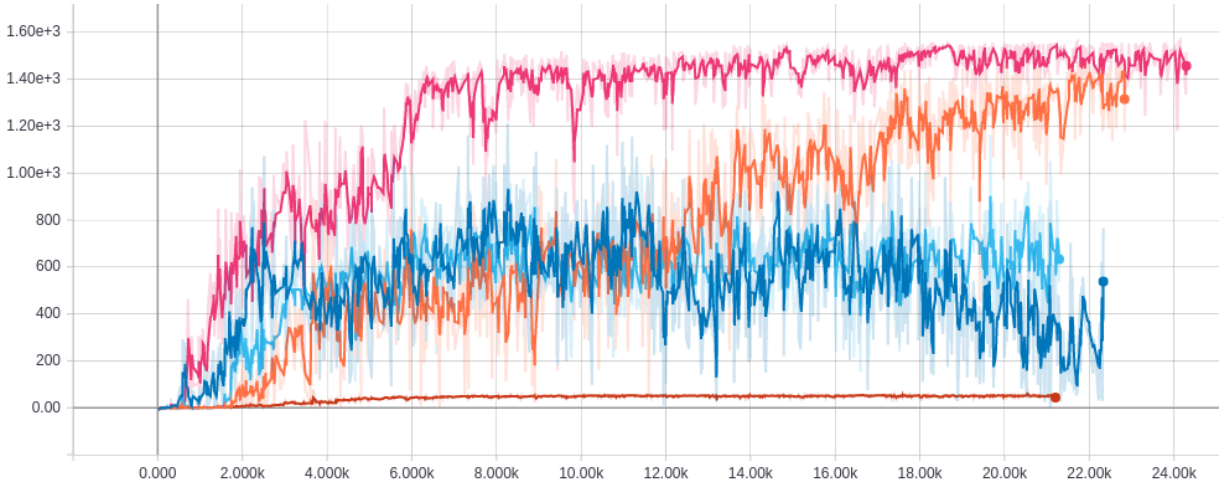
but could not make further progress. The red case never learned to complete the task, and the episodes ran to the maximum steps 500.

5.4.3.3 *One sparse reward (one-sparse)*

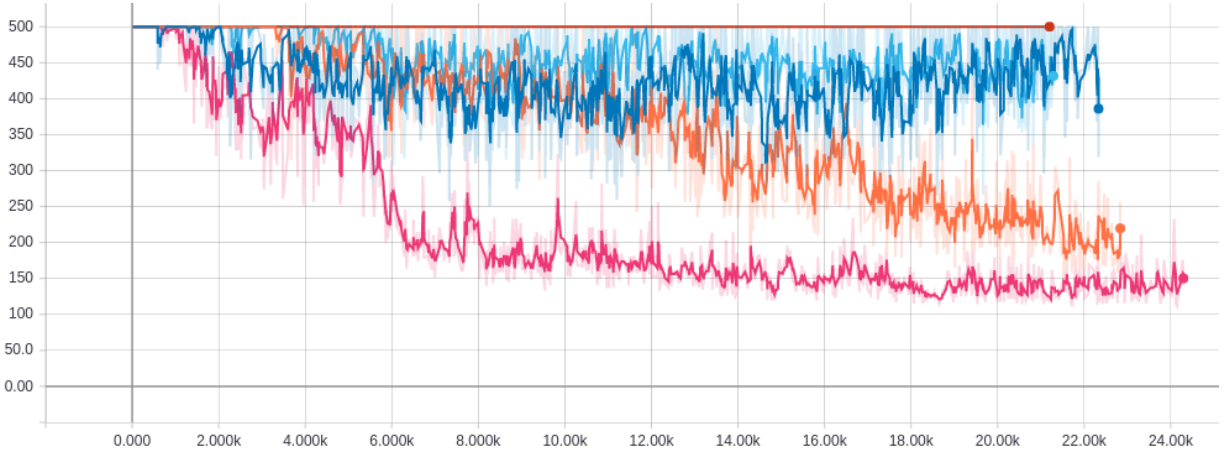
Fig. 5.11 shows the mean episode reward and mean episode length during the training with five different random seeds in the reward scheme of *one sparse reward* (Eq. 5.10). Different colors mean different random seeds. The model was trained for 23,000 iterations at most (2 days and 12 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training are 57.5M at most. None of the models could learn even to make the gripper close to the tool. It seems that one of the episode was lucky to complete the task (the high spike in the red learning curve) at around 900 iterations, but the one lucky trial could not facilitate further progress. Fig. 5.11a shows the mean episode reward during training. Obviously, none of the models could get reward (except the red case that got lucky). The agents learned to reduce the torque penalty, but could not learn any of the subtasks. Fig. 5.11b shows the mean episode length during training. Obviously, none of them learned to complete the task so that all the episodes lasted to the maximum step of 500.

5.4.3.4 *All reward elements without stepwise order (no-stepwise-all)*

Fig. 5.12 shows the mean episode reward and mean episode length during training with five different random seeds in the reward scheme of *all reward elements without stepwise order* (Eq. 5.11). Different colors mean different random seeds. The model was trained for 22,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training were 55M at most. Since all the rewards elements are always provided in this scheme, the models always show some amounts of reward, but they are all fluctuating between around 700 and 850, and none of them showed progress. Interestingly, none of them could not even learn to move the gripper close to the tool even though the continuous intermediate reward element for the distance between the gripper and the tool is

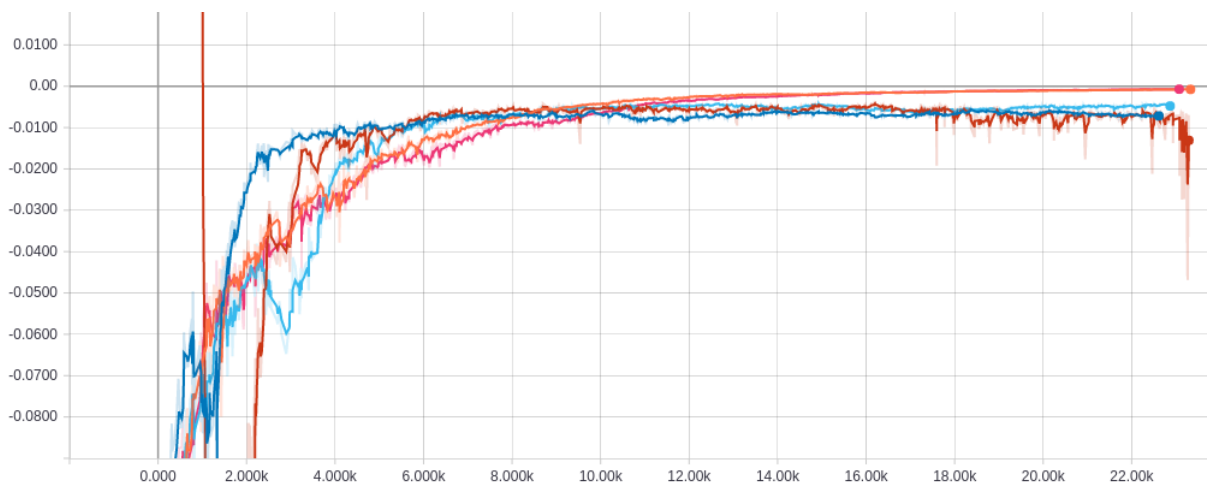


(a) Mean episode reward

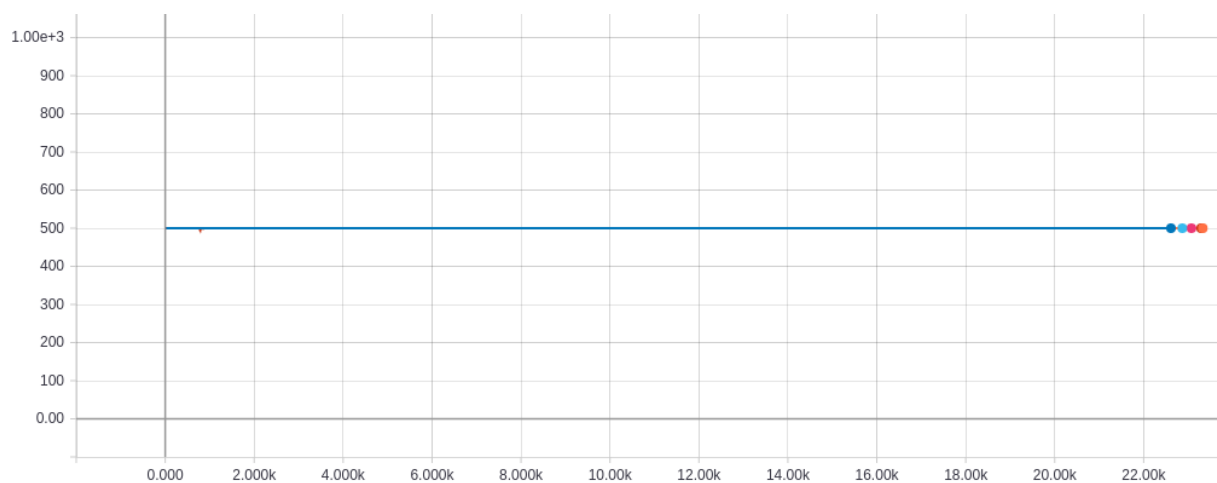


(b) Mean episode length

Figure 5.10: Learning curve for different random seeds under *stepwise composite reward without intermediate elements (stepwise-no-intermediate)*: Mean episode reward and mean episode length during training with five different random seeds. Different colors mean different random seeds. The model instances were trained for 24,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training are 60M at most. The pink case successfully learned the task at around 7,000 iterations. The orange case learned gradually and reached the successful learning state at around 23,000 iterations. The others could not learn the task. The blue and sky-blue cases could not get out of the plateau, where they could learn the first two subtasks (i.e. Reach-Tool and Grasp-Tool), but could not learn to the other two subtasks (i.e. Tool-Reach-Object and Object-Reach-Target). The red case could not show much learning, where it could learn to make the gripper close to the tool handle, but closes the gripper too early so that it was not able to grasp the tool. a) Mean episode reward during training. (b) Mean episode length during training. The graph shows inverse correlation with the graph in (a). See text for details.



(a) Mean episode reward



(b) Mean episode length

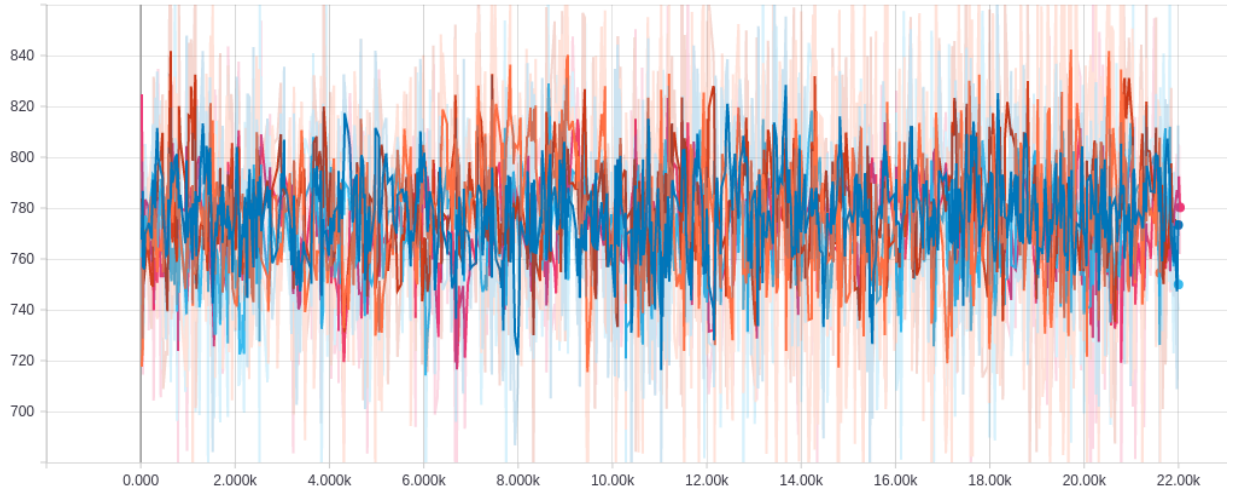
Figure 5.11: Learning curve for different random seeds under *one sparse reward (one-sparse)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 23,000 iterations at most (2 days and 12 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training are 57.5M at most. None of the models could learn even to make the gripper close to the tool. It seems that one of the episode was lucky to complete the task (the high spike in the red learning curve) at around 900 iterations, but the one lucky trial could not facilitate further progress. (a) Mean episode reward during training. Obviously, none of the models could get reward (except the red case that got lucky). The agents learned to reduce the torque penalty, but could not learn any of the subtasks. (b) Mean episode length during training. Obviously, none of them learned to complete the task so that all the episodes lasted to the maximum step of 500. See text for details.

provided. This indicates that presenting extra rewards such as the distance between the tool and the object and/or the one between the object and the target before learning the first subtasks (reach and grab) affects the learning negatively. In general, rewards unnecessary for learning the current subtask become noise, which hinders proper learning. This demonstrates the difficulty of shaping rewards for task consisting of many sequential subtasks such as tool-use. Fig. 5.12a shows the mean episode reward during training. All rewards fluctuated between about 700 and 850. None of them learned the first subtask (Reach-Tool). Fig. 5.12b shows the mean episode length during training. None of them completed the task before reaching the maximum time step of 500.

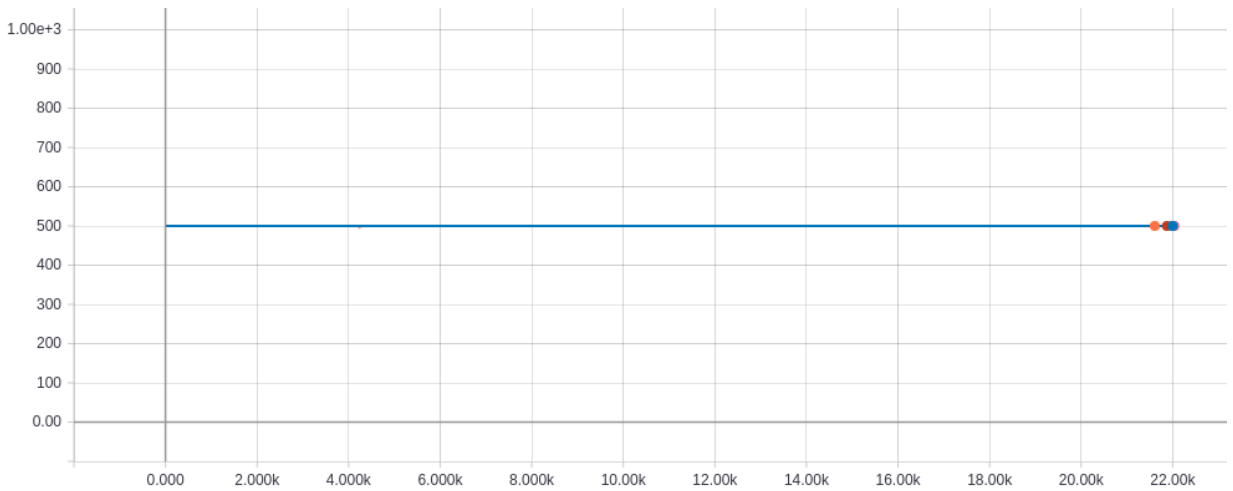
5.5 Discussion

Given the gap of the tool-use tasks in AI and Robotics and in deep RL literature, in this chapter, a more complex tool-use environment is implemented in a physics based simulator. A contact-rich tool-use environment was proposed and implemented in a physics based simulator, where four subtasks need to be completed in a specific order in continuous action spaces. This tool-use environment can be easily used further deep RL algorithm development and benchmarks. Also, it is demonstrated that the complex tool-use task consisting of multiple sequential subtasks can be successfully learned with stepwise composite reward shaping using a deep reinforcement learning method (ACKTR). It is shown that without stepwise reward shaping, the intermediate continuous reward elements alone cannot make the tool-use learning progress although they include all distance information to the subtasks completions. The difficulty of the tool-use task can be easily adjusted by selecting one of the four reward schemes, which could be utilized for further deep RL algorithm development as well.

We learned that designing contact-rich task environment in physics based simulator is not trivial. We needed to make feasible task environment. There were issues such as penetration



(a) Mean episode reward



(b) Mean episode length

Figure 5.12: Learning curve for different random seeds under *all reward elements without stepwise order (no-stepwise-all)*: Mean episode reward and mean episode length during the training with five different random seeds. Different colors mean different random seeds. The model was trained for 22,000 iterations at most (2 days and 4 hours). One iteration consisted of 2,500 steps, so the total steps taken for the training were 55M at most. Since all the rewards elements are always provided in this scheme, the models always show some amounts of reward, but they are all fluctuating between around 700 and 850, and none of them showed progress. Interestingly, none of them could not even learn to move the gripper close to the tool even though the continuous reward element for the distance between the gripper and the tool is provided. This indicate that presenting extra rewards such as the distance between the tool and the object and/or the one between the object and the target before learning the first subtask (reach) affects the learning negatively. (a) Mean episode reward during the training. (b) Mean episode length during the training. None of them learned the first subtask (Reach-Tool) and completed the task before reaching the maximum time step of 500. See text for details.

and slip between objects, so that the attributes of the objects including scale, surface, mass, and maximum joint force needed to be carefully designed. Besides, it was difficult to know whether the learning fails due to the task environment or algorithm.

We also learned that reward shaping for tool-use is not trivial. Intermediate continuous reward guides the learning with continuous reward signals, but a continuous reward signal for a subtask can interrupt another subtask learning. Stepwise reward shaping helps in this case. We found that if stepwise reward is well shaped, intermediate continuous reward is not essential for the tool-use learning.

In addition to stepwise reward shaping, hierarchical (or meta) learning or imitation learning approaches can be used for tool-use learning. In hierarchical learning [8, 66, 39, 41], lower hierarchies learn subskills and higher hierarchies learn choosing an appropriate subskill given a state. However, designing hierarchies for subskills could be complex as well. To mitigate the effort for hierarchy design, automatic discoveries of subskills were demonstrated for simple tasks (e.g. ant walking in four directions) in the recent years [41]. In imitation learning [6, 105, 38, 55], expert demonstrations from humans, joystick control, kinesthetic teaching, planning/control algorithm, etc. are given. Agents try to mimic the trajectories from the demonstrations. However, imitation learning needs many demonstrations and collecting good trajectories is not trivial. Also mapping demonstrated trajectories from humans to agents' body spaces is complex. By combining stepwise reward shaping with hierarchical learning and imitation learning, more complex tool-use tasks could be learned.

For the future work, we consider a deep NN (RL) and neuroevolution combined approach. Despite the recent deep RL achievement in many challenging control tasks, some degree of human knowledge such as reward shaping, human demonstrations, hierarchy design, hyperparameter turning, etc. is still required to learn complex tool-use tasks. Also, deep RL tends to converge to one sub-optimal solution, but it is necessary to try diverse solutions to solve more complex tool-use tasks such as tool construction. Neuroevolution and deep neural

network combined approaches have been recently proposed for control tasks in continuous spaces [31, 122, 109]. These approaches minimize or do not require the human knowledge mentioned above. In addition, the above neuroevolution approaches have demonstrated to be able to discover a variety of different ways to solve problems than deep RL algorithms. The neuroevolution approaches used a large number of CPUs (e.g. 700+ CPU) to run a large number of populations, which is essential for learning challenging tasks. Although the neuroevolution approaches above have shown good performance on the typical deep RL benchmarks, complex tasks such as tool-use have not yet been performed. Therefore, we consider neuroevolution and deep NN (RL) combined approaches for the future research to attack more complex tool-use tasks such as tool construction with minimal human knowledge.

5.6 Conclusion

Given the gap of the tool-use tasks in AI and Robotics and in deep RL literature, in this chapter, a more complex tool-use environment is implemented in a physics simulator. Also, it is demonstrated that the complex tool-use task consisting of multiple sequential subtasks can be successfully learned with stepwise composite reward shaping using a deep reinforcement learning method. First, a contact-rich tool-use environment is proposed and implemented in a physics based simulator. In the tool-use environment, four subtasks are required to be completed in a specific order in continuous action spaces. The goal of the task is to drag a distant object to target locations using a T-shaped tool by controlling an articulated arm (three joints) with a gripper (one joint). Second, it is demonstrated that the complex tool-use task consisting of multiple sequential subtasks can be successfully learned with stepwise composite reward shaping using a deep reinforcement learning method (ACKTR). The stepwise composite reward schemes are compared to the other two baseline reward schemes to show the effect of stepwise composite reward shaping. It is shown that

without stepwise reward shaping, the intermediate continuous reward elements alone cannot make the tool-use learning progress although they include all distance information to the subtasks completions. While the intermediate continuous reward can guide the learning with continuous reward signals, we found that a continuous reward signal for a subtask can interfere with another subtask learning. In this case, stepwise reward shaping helps. We found that if a stepwise reward is well shaped, an intermediate continuous reward is not essential for the tool-use learning. Overall, the tool-use task environment and the findings with respect to reward shaping in this chapter can be utilized for further deep RL algorithm development and benchmarks, which could facilitate the development of sensorimotor agents capable of using tools in the real world.

6. DISCUSSION AND CONCLUSION

Sensorimotor learning is an essential function of the brain, whether biological or artificial, to adapt to the changing environment throughout the life of the agent. However, there is still a lack of understanding about the mechanisms of sensorimotor learning in the brain. In this dissertation, I investigated three topics to probe the nature of sensorimotor aspects of learning in the brain, through computational modeling.

- (1) Development: Self-organization of the motor map in the cortex.
- (2) Internal Dynamics: Predictable internal brain dynamics and its role in authorship of actions.
- (3) Tool-Use with Neuroevolution Controller and with Deep Reinforcement Learning.

The summary and contributions for each topic are as follows.

(1) A sensorimotor agent needs to understand its own body, a map of its own behavior. Motivated by an experimental study [47] showing a topographical map of complex behaviors in the macaque brain, we developed a target reaching gesture map using a biologically motivated self-organizing map model of the cortex (GCAL model, a simplified yet enhanced version of the LISSOM model) with two-joint arm movements as input. The resulting gesture map showed a global topographic order based on the target locations. The map was comparable to the motor map reported in the experimental study [47]. Our work is an important first step toward a fully motor-based motor map development, and I expect the findings reported in this work to help us better understand the general nature of cortical map development in a sensorimotor agent. (Chapter 2)

(2) Previous computer simulation studies showed that neural network controllers with more predictable internal state dynamics can attain higher performance in harsher or chang-

ing environments, which also indicates higher chance of survival in evolution [68, 29, 25]. Such predictable internal state dynamics could be the basis of prediction of external behavior and authorship of actions. This finding is profound since it implies that the internal state's properties can affect the external behavioral performance especially in changing environment, and predictable internal state dynamics could be a necessarily condition for intelligent agents in the evolutionary pathway to have authorship of actions and to adapt themselves to changing environments. However, there was a missing link to connect between the findings from the simulation studies and the brain. The findings from the simulation study do not necessarily mean that the internal state dynamics affects the external behavior (authorship of actions) in the biological brain as well. To fill this gap, we investigated the role of predictability of internal state dynamics in the brain by analyzing the human EEG data with conscious state as a surrogate of authorship of actions. We analyzed and predicted inter-peak interval (IPI) in the internal brain dynamics data (EEG) and discussed the relation between predictable internal brain dynamics and intelligent behaviors (sensorimotor actions). The results show that, for all four subjects, on average, both awake state and REM have lower IPI prediction error than SWS. These results support our hypothesis regarding the predictability of internal state dynamics and conscious states (i.e., they should be correlated). Our findings show that the existence of predictable dynamics in the brain and its relation to conscious states (as a surrogate of authorship of actions), thereby filling the missing connection between the previous simulation studies [68, 29, 25] and the biological brain. The findings implicate that the predictable internal state dynamics could be a necessarily condition for intelligent sensorimotor agents in the evolutionary pathway to have authorship of actions and to adapt themselves to changing environment. A deeper understanding of this connection between predictable internal dynamics and external behaviors (authorship of actions) can lead to robust control mechanisms in intelligent sensorimotor agents. (Chapter 3).

(3) Tool-use is a salient indicator of intelligence that requires high levels of sensorimotor

skill learning and problem solving capabilities. However, tool-use is still largely under-explored in the field of AI including the DL literature. In this topic, I present environments and sensorimotor agents where the agents can adapt to use simple or complex tools based on minimal task knowledge. Specifically, I present a two step approach where I first evolve neural network controllers for simple tool-use in reaching tasks with minimal task knowledge followed by analysis of the evolved networks, and then implement a more complex tool-use task such as dragging an object to a target location using a tool in a physics simulation environment and demonstrate that deep RL together with composite reward shaping can learn the complex tool-use task successfully.

In the first step (Chapter 4), we used a neural circuit evolution algorithm that permits incrementally changing topology (NEAT) to evolve a neural controller for a two degree-of-freedom articulated limb in target-reaching task. Our results show that minimal, indirect fitness criteria such as distance to goal, speed to reach the goal, and tool pick-up frequency are enough to give rise to tool-use behavior. We also found that among the successful neural circuit controllers, those with more recurrent loops were even more effective. Also, we presented some analysis of the internal dynamics of evolved neural circuits and correlate the dynamics with behavior.

In the second step (Chapter 5), a more complex tool-use environment is implemented in a physics simulator. Also, it is demonstrated that complex tool-use task consisting of multiple sequential subtasks can be successfully learned with stepwise composite reward shaping using a cutting edge deep reinforcement learning method. This part makes two contributions. First, the tool-use environment is presented that consists of four subtasks to be completed in specific order. The goal of the task is to drag a distant object to the target location using a T-shaped tool by controlling an articulated arm (three joints) with a gripper (one joint). To complete the task successfully, the gripper needs to reach the tool handle, grab the tool, move the tool end-effector to the object, and then finally move

the object to the target location using the tool. The task is implemented in the MuJoCo physics simulator for continuous action space control with OpenAI Gym toolkit. This tool-use task environment could be easily used for further deep RL algorithm development and benchmarks. Second, stepwise composite reward shaping is investigated for learning the multiple subtasks in specific order in the tool-use task. It is demonstrated that the tool-use task can be successfully learned using stepwise composite reward shaping with a deep RL algorithm called Actor Critic using Kronecker-Factored Trust Region (ACKTR). The stepwise composite reward scheme is compared to the other two baseline reward schemes to show the effect of stepwise composite reward shaping. It is shown that the reward function without stepwise organization could not make progress in learning even though all the reward elements including continuous intermediate elements are presented. Also, the experiments show that the difficulty of the tool-use task can be easily controlled by selecting one of the four reward schemes, which could be utilized for further deep RL algorithm development and benchmarks.

Overall, the studies in this dissertation are expected to help us understand the nature of sensorimotor aspects of learning in the brain and even implement them in AI. Also, the simulation environments and codes used in this dissertation could benefit various future research in cortical motor map development and in algorithm development for tool-use.

REFERENCES

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Tyson N. Aflalo and Michael S. A. Graziano. Possible origins of the complex topographic organization of motor cortex: Reduction of a multidimensional space onto a two-dimensional array. *The Journal of Neuroscience*, 26:6288–6297, 2006.
- [3] Adrian Agogino, Kagan Tumer, and Risto Miikkulainen. Efficient credit assignment through evaluation function decomposition. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1309–1316. ACM, 2005.
- [4] Colin Allen. Transitive inference in animals: Reasoning or conditioned associations. *Rational animals*, pages 175–185, 2006.
- [5] Robert St Amant and Alexander B Wood. Tool use for autonomous agents. In *Twentieth AAAI Conference on Artificial Intelligence*, pages 184–189, 2005.
- [6] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

- [7] Artur M Arsenio. Learning task sequences from scratch: applications to the control of tools and toys by a humanoid robot. In *Control Applications, 2004. Proceedings of the 2004 IEEE International Conference on*, volume 1, pages 400–405. IEEE, 2004.
- [8] Kai Arulkumaran, Nat Dilokthanakul, Murray Shanahan, and Anil Anthony Bharath. Classifying options for deep reinforcement learning. *arXiv preprint arXiv:1604.08153*, 2016.
- [9] AMI Auersperg, AMI von Bayern, S Weber, A Szabadvari, T Bugnyar, and A Kacelnik. Social transmission of tool use and tool manufacture in goffin cockatoos (*catua goffini*). *Proceedings of the Royal Society of London B: Biological Sciences*, 281(1793):20140972, 2014.
- [10] Bram Bakker and Michiel De Jong. The epsilon state count. *From Animals to Animats*, 6:51–60, 2000.
- [11] Domna Banakou and Mel Slater. Body ownership causes illusory self-attribution of speaking and influences subsequent real speaking. *Proceedings of the National Academy of Sciences*, 111(49):17678–17683, 2014.
- [12] James A. Bednar. Building a mechanistic model of the development and function of the primary visual cortex. *Journal of Physiology-Paris*, 106(5-6):194–211, 2012.
- [13] Randall D Beer. A dynamical systems perspective on agent-environment interaction. *Artificial intelligence*, 72(1-2):173–215, 1995.
- [14] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.
- [15] GH Bishop. Feedback through the environment as an analog of brain functioning. *Self-Organizing Systems, MC Yovitts & S. Cameron, Eds*, pages 122–152, 1960.

- [16] Sarah-Jayne Blakemore and Chris Frith. Self-awareness and action. *Current opinion in neurobiology*, 13(2):219–224, 2003.
- [17] KJ Blinowska and M Malinowski. Non-linear and linear forecasting of the eeg time series. *Biological cybernetics*, 66(2):159–165, 1991.
- [18] Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.
- [19] Ch Boesch and Hedwige Boesch. Tool use and tool making in wild chimpanzees. *Folia primatologica*, 54(1-2):86–99, 1990.
- [20] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [21] Solly Brown and Claude Sammut. Tool use and learning in robots. In *Encyclopedia of the Sciences of Learning*, pages 3327–3330. Springer, 2012.
- [22] Daniel Bullock, Stephen Grossberg, and Frank H Guenther. A self-organizing neural model of motor equivalent reaching and tool use by a multijoint arm. *Journal of Cognitive Neuroscience*, 5(4):408–435, 1993.
- [23] Corrado Cavallero, Piercarla Cicogna, Vincenzo Natale, Miranda Occhionero, et al. Slow wave sleep dreaming. *Sleep: Journal of Sleep Research & Sleep Medicine*, 1992.
- [24] Yoonsuck Choe. Connectome, general. *Encyclopedia of Computational Neuroscience*, 2014.
- [25] Yoonsuck Choe, Jaerock Kwon, and Ji Ryang Chung. Time, consciousness, and mind uploading. *International Journal on Machine Consciousness*, 4:257–274, 2012.

- [26] Yoonsuck Choe, Qinbo Li, Jin Huang, and Jaewook Yoo. Synthetic connectomics to tackle big data challenges in its natural counterpart. In *US-Korea Conference on Science, Technology, and Entrepreneurship (UKC 2015)*, 2015.
- [27] Yoonsuck Choe, Jaewook Yoo, and Qinbo Li. Tool construction and use challenge: Tooling test rebooted. In *AAAI-15 Workshop on Beyond the Turing Test*, 2 pages, 2015.
- [28] Ji Ryang Chung and Yoonsuck Choe. Emergence of memory in reactive agents equipped with environmental markers. *Autonomous Mental Development, IEEE Transactions on*, 3(3):257–271, 2011.
- [29] Ji Ryang Chung, Jaerock Kwon, Timothy A. Mann, and Yoonsuck Choe. Evolution of time in neural networks: From the present to the past, and forward to the future. In A. Ravishankar Rao and Guillermo A. Cecchi, editors, *The Relevance of the Time Domain to Neural Network Models, Springer Series in Cognitive and Neural Systems 3*, pages 99–116. Springer, New York, 2012.
- [30] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- [31] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.
- [32] Damien Coyle, Girijesh Prasad, and Thomas Martin McGinnity. A time-series prediction approach for feature extraction in a brain-computer interface. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 13(4):461–467, 2005.

- [33] A. Delorme and S. Makeig. EEGLAB: An open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods*, 134:9–21, 2004.
- [34] Prafulla Dhariwal, Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. <https://github.com/openai/baselines>, 2017.
- [35] Ezequiel A Di Paolo. Organismically-inspired robotics: homeostatic adaptation and teleology beyond the closed sensorimotor loop. *Dynamical systems approach to embodiment and sociality*, pages 19–42, 2003.
- [36] S. Droit-Volet and Pierre S. Zélanti. Development of time sensitivity and information processing speed. *PLoS One*, 8:e71424, 2013.
- [37] Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.
- [38] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.
- [39] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.
- [40] Preston Foerder, Marie Galloway, Tony Barthel, Donald E Moore III, and Diana Reiss. Insightful problem solving in an asian elephant. *PloS one*, 6(8):e23251, 2011.
- [41] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

- [42] Brad Fullmer and Risto Miikkulainen. Using marker-based genetic encoding of neural networks to evolve finite-state behaviour. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 255–262. MIT Press, 1992.
- [43] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.
- [44] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317–342, 1997.
- [45] Afonso Gonçalves, João Abrantes, Giovanni Saponaro, Lorenzo Jamone, and Alexandre Bernardino. Learning intermediate object affordances: Towards the development of a tool concept. In *Development and Learning and Epigenetic Robotics (ICDL-Epirob), 2014 Joint IEEE International Conferences on*, pages 482–488. IEEE, 2014.
- [46] Michael S.A. Graziano, Charlotte S.R. Taylor, and Tirin Moore. Complex movements evoked by microstimulation of precentral cortex. *Neuron*, 34:841–851, 2002.
- [47] Michael SA Graziano, Charlotte SR Taylor, and Tirin Moore. Complex movements evoked by microstimulation of precentral cortex. *Neuron*, 34(5):841–851, 2002.
- [48] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.
- [49] Michael D Gumert, Marius Kluck, and Suchinda Malaivijitnond. The physical characteristics and usage patterns of stone axe and pounding hammers used by long-tailed macaques in the andaman sea region of thailand. *American Journal of Primatology*, 71(7):594–608, 2009.

- [50] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the Marquadt algorithm. *IEEE Transactions on Neural Networks*, 5:989–993, 1994.
- [51] Patrick Haggard, Sam Clark, and Jeri Kalogeras. Voluntary action and conscious awareness. *Nature neuroscience*, 5(4):382, 2002.
- [52] K Ronald L Hall. Tool-using performances as indicators of behavioral adaptability. *Current Anthropology*, 4(5):479–494, 1963.
- [53] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.
- [54] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI (Accepted)*, 2018.
- [55] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. *arXiv preprint arXiv:1704.03732*, 2017.
- [56] Martin Hülse, Steffen Wischmann, Poramate Manoonpong, Arndt von Twickel, and Frank Pasemann. Dynamical systems in the sensorimotor loop: On the interrelation between internal and external mechanisms of evolved robot behavior. In *50 years of artificial intelligence*, pages 186–195. Springer, 2007.
- [57] Raoul Huys, Breanna E. Studenka, Nicole L. Rheaume, Howard N. Zelaznik, and Viktor K. Jirsa. Distinct timing mechanisms produce discrete and continuous movements. *PLoS Comput Biol*, 4:e1000061, 2008.
- [58] Raghvendra Jain and Tetsunari Inamura. Learning of usage of tools based on interaction between humans and robots. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, pages 597–602. IEEE, 2014.

- [59] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247, 2016.
- [60] Dov Katz and Oliver Brock. Manipulating articulated objects with interactive perception. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 272–277. IEEE, 2008.
- [61] M. Kawato. Internal models for motor control and trajectory planning. *Current Opinions on Neurobiology*, 9:718–727, 1999.
- [62] Bob Kemp, Aeilko H Zwinderman, Bert Tuk, Hilbert AC Kamphuisen, and Josefien JL Obery. Analysis of a sleep-dependent neuronal feedback loop: the slow-wave micro-continuity of the eeg. *Biomedical Engineering, IEEE Transactions on*, 47(9):1185–1194, 2000.
- [63] Ben Kenward, Alex AS Weir, Christian Rutz, and Alex Kacelnik. Behavioural ecology: Tool manufacture by naive juvenile crows. *Nature*, 433(7022):121–121, 2005.
- [64] Wolfgang Köhler. *The mentality of apes*. Read Books Ltd, 2013.
- [65] T. Kohonen. *Self-organizing maps*. Springer, 3rd edition, 2001.
- [66] 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.
- [67] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- [68] Jaerock Kwon and Yoonsuck Choe. Internal state predictability as an evolutionary precursor of self-awareness and agency. In *Proceedings of the Seventh International Conference on Development and Learning*, pages 109–114. IEEE, 2008.

- [69] Jaerock Kwon and Yoonsuck Choe. Predictive internal neural dynamics for delay compensation. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 443–448. IEEE, 2010.
- [70] Judith S. Law, Jan Antolik, and James A. Bednar. Mechanisms for stable and robust development of orientation maps and receptive fields. Technical report, School of Informatics, The University of Edinburgh, EDI-INF-RR-1404., 2011.
- [71] Dongheui Lee, Hirotohi Kunori, and Yoshihiko Nakamura. Association of whole body motion from tool knowledge for humanoid robots. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2867–2874. IEEE, 2008.
- [72] 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. In *ICLR*, volume 2016, 2016.
- [73] Heejin Lim and Yoonsuck Choe. Compensating for neural transmission delay using extrapolatory neural activation in evolutionary neural networks. *Neural Information Processing—Letters and Reviews*, 10:147–161, 2006.
- [74] Michail Maniadakis, Marc Wittmann, and Panos Trahanias. Time experiencing by robotic agents. In *ESANN 2011 Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 27–29, 2011.
- [75] Timothy A. Mann and Yoonsuck Choe. Neural conduction delay forces the emergence of predictive function in an evolving simulation. *BMC Neuroscience*, 11(Suppl 1):P62, 2010. Nineteenth Annual Computational Neuroscience Meeting: CNS*2010.
- [76] Timothy A Mann and Yoonsuck Choe. Prenatal to postnatal transfer of motor skills through motor-compatible sensory representations. In *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*, pages 185–190. IEEE, 2010.

- [77] Tanis Mar, Vadim Tikhanoff, Giorgio Metta, and Lorenzo Natale. Self-supervised learning of grasp dependent tool affordances on the icub humanoid robot. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3200–3206. IEEE, 2015.
- [78] Angelo Maravita and Atsushi Iriki. Tools for the body (schema). *Trends in cognitive sciences*, 8(2):79–86, 2004.
- [79] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.
- [80] J. McIntyre, M. Zago, A. Berthoz, and F. Lacquaniti. Does the brain model Newton’s laws? *Nature Neuroscience*, 4:693–694, 2001.
- [81] Risto Miikkulainen, James A. Bednar, Yoonsuck Choe, and Joseph Sirosh. *Computational Maps in the Visual Cortex*. Springer, 2005.
- [82] 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.
- [83] 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.
- [84] Felicitas Mokom and Ziad Kobti. Evolution of artifact capabilities. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 476–483. IEEE, 2011.
- [85] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

- [86] David E Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1997.
- [87] F. C. Nather, J. L. O. Bueno, E. Bigand, and S. Droit-Volet. Time changes with the embodiment of another’s body posture. *PLoS ONE*, 6:e19818, 2011.
- [88] Shun Nishide, Jun Tani, Toru Takahashi, Hiroshi G Okuno, and Tetsuya Ogata. Tool–body assimilation of humanoid robot using a neurodynamical system. *Autonomous Mental Development, IEEE Transactions on*, 4(2):139–149, 2012.
- [89] Kazuo Okanoya, Naoko Tokimoto, Noriko Kumazawa, Sayaka Hihara, and Atsushi Iriki. Tool-use training in a species of rodent: the emergence of an optimal motor strategy and functional understanding. *PloS one*, 3(3):e1860, 2008.
- [90] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Agens faber: Toward a theory of artefacts for mas. *Electronic Notes in Theoretical Computer Science*, 150(3):21–36, 2006.
- [91] Choonseog Park, Yoon H. Bai, and Yoonsuck Choe. Tactile or visual?: Stimulus characteristics determine receptive field type in a self-organizing map model of cortical development. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing*, pages 6–13, 2009.
- [92] Frank Pasemann. Neurodynamics in the sensorimotor loop: Representing behavior relevant external situations. *Frontiers in Neurorobotics*, 11, 2017.
- [93] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA ’09. IEEE International Conference on*, pages 763–768. IEEE, 2009.
- [94] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

- [95] David Philipona, Jk O'regan, Jean-Pierre Nadal, and Olivier JMD Coenen. Perception of the structure of the physical world using unknown multimodal sensors and effectors. In *NIPS*, volume 16, page 17, 2003.
- [96] Iyaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.
- [97] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.
- [98] Elizabeth E Price, Susan P Lambeth, Steve J Schapiro, and Andrew Whiten. A potent effect of observational learning on chimpanzee tool construction. *Proceedings of the Royal Society of London B: Biological Sciences*, 276(1671):3377–3383, 2009.
- [99] Jose C Principe, Alok Rathie, and Jyh-Ming Kuo. Prediction of chaotic time series with neural networks and the issue of dynamic modeling. *International Journal of Bifurcation and Chaos*, 2(04):989–996, 1992.
- [100] Tomos Proffitt, Lydia V Luncz, Tiago Falótico, Eduardo B Ottoni, Ignacio de la Torre, and Michael Haslam. Wild monkeys flake stone tools. *Nature*, 2016.
- [101] Randall Reams, Jaewook Yoo, and Yoonsuck Choe. Emergence of tool construction in an articulated limb controlled by evolved neural circuits. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017.
- [102] Mark Ring and Tom Schaul. The organization of behavior into temporal and spatial neighborhoods. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–6, 2012.

- [103] Mark Ring, Tom Schaul, and Juergen Schmidhuber. The two-dimensional organization of behavior. In *2011 IEEE International Conference on Development and Learning (ICDL)*, pages 1–8, 2011.
- [104] Robert Rosen, J Rosen, J Kineman, and M Nadin. Anticipatory systems: Philosophical, mathematical, and methodological foundations (ifsr international series on systems science and engineering), 1985.
- [105] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.
- [106] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [107] Eytan Ruppin. Evolutionary autonomous agents: A neuroscience perspective, 2002.
- [108] Ryo Saegusa, Giorgio Metta, Giulio Sandini, and Lorenzo Natale. Developmental perception of the self and action. *IEEE transactions on neural networks and learning systems*, 25(1):183–202, 2014.
- [109] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- [110] Bulcsú Sándor, Tim Jahn, Laura Martin, and Claudius Gros. The sensorimotor loop as a dynamical system: How regular motion primitives may emerge from self-organized limit cycles. *arXiv preprint arXiv:1511.04338*, 2015.
- [111] Boris Schäfer, Nicklas Bergfeldt, María José Riveiro Carballa, and Tom Ziemke. Evolution of tool use behavior. In *Proceedings of the First IEEE Symposium on Artificial Life*, pages 31–38. Citeseer, 2007.

- [112] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.
- [113] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [114] Robert W Shumaker, Kristina R Walkup, and Benjamin B Beck. *Animal tool behavior: the use and manufacture of tools by animals*. JHU Press, 2011.
- [115] 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):484–489, 2016.
- [116] Rachel Smolker, Andrew Richards, Richard Connor, Janet Mann, and Per Berggren. Sponge carrying by dolphins (delphinidae, tursiops sp.): a foraging specialization involving tool use? *Ethology*, 103(6):454–465, 1997.
- [117] Charles T Snowdon. Language capacities of nonhuman animals. *American Journal of Physical Anthropology*, 33(S11):215–243, 1990.
- [118] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [119] M. Steriade, I. Timofeev, and F. Grenier. Natural waking and sleep states: A view from inside neocortical neurons. *Journal of Neurophysiology*, 85:1969–1985, 2001.
- [120] Alexander Stoytchev. Behavior-grounded representation of tool affordances. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3060–3065. IEEE, 2005.

- [121] Jörg Stückler and Sven Behnke. Adaptive tool-use strategies for anthropomorphic service robots. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 755–760. IEEE, 2014.
- [122] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.
- [123] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.
- [124] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [125] Richard S. Sutton and Andrew G. Barto. Reinforcement learning : An introduction second edition , in progress. 2017.
- [126] Kuniyuki Takahshi, Tetsuya Ogata, Hadi Tjandra, Yuki Yamaguchi, Yuki Suga, and Shigeki Sugano. Tool-body assimilation model using a neuro-dynamical system for acquiring representation of tool function and motion. In *Advanced Intelligent Mechatronics (AIM), 2014 IEEE/ASME International Conference on*, pages 1255–1260. IEEE, 2014.
- [127] Andreas Thiel, Helmut Schwegler, and Christian W. Eurich. Complex dynamics is abolished in delayed recurrent systems with distributed feedback times. *Complexity*, 8:102–108, 2003.
- [128] Serge Thill, Daniele Caligiore, Anna M Borghi, Tom Ziemke, and Gianluca Baldassarre. Theories and computational models of affordance and mirror systems: an integrative review. *Neuroscience & Biobehavioral Reviews*, 37(3):491–521, 2013.

- [129] V Tikhanoff, U Pattacini, L Natale, and G Metta. Exploring affordances and tool use on the icub. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 130–137. IEEE, 2013.
- [130] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.
- [131] Fay S. Tyner and John R. Knott. *Fundamentals of EEG Technology: Volume 1 Basic concepts and methods*. Lippincott Williams & Wilkins, Philadelphia, PA, 1983.
- [132] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.
- [133] Elisabetta Visalberghi and Luca Limongelli. Acting and understanding: Tool use revisited through the minds of capuchin monkeys. *Reaching into thought: The minds of the great apes*, pages 57–79, 1996.
- [134] Julien Vitay and Fred H Hamker. A computational model of basal ganglia and its role in memory retrieval in rewarded visual memory tasks. *Frontiers in Computational Neuroscience*, 4(13), 2010.
- [135] Liyu Wang, Luzius Brodbeck, and Fumiya Iida. Mechanics and energetics in tool manufacture and use: a synthetic approach. *Journal of the Royal Society Interface*, 11(100):20140827, 2014.
- [136] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.
- [137] Steven D Whitehead and Dana H Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

- [138] Andrew Whiten, Jane Goodall, William C McGrew, Toshisada Nishida, Vernon Reynolds, Yukimaru Sugiyama, Caroline EG Tutin, Richard W Wrangham, and Christophe Boesch. Cultures in chimpanzees. *Nature*, 399(6737):682–685, 1999.
- [139] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W Anderson. Genetic reinforcement learning for neurocontrol problems. In *Genetic Algorithms for Machine Learning*, pages 103–128. Springer, 1993.
- [140] Alexis P Wieland. Evolving controls for unstable systems. In *Connectionist models: proceedings of the 1990 summer school*, pages 91–102, 1990.
- [141] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
- [142] Stuart P. Wilson, Judith S. Law, Ben Mitchinson, Tony J. Prescott, and James A. Bednar. Modeling the emergence of whisker direction maps in rat barrel cortex. *PLoS One*, 5(1), 2010.
- [143] Marc Wittmann. The inner sense of time: how the brain creates a representation of duration. *Nature Reviews Neuroscience*, 14:217–223, 2013.
- [144] D. M. Wolpert and J. R. Flanagan. Motor prediction. *Current Biology*, 11:R729–R732, 2001.
- [145] D. M. Wolpert, Z. Ghahramani, and M. I. Jordan. An internal model for sensorimotor integration. *Science*, 269:1880–1882, 1995.
- [146] D. M. Wolpert, R. C. Miall, and M. Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2:338–347, 1998.
- [147] Alexander B Wood, Thomas E Horton, and R St Amant. Effective tool use in a habile agent. In *Systems and Information Engineering Design Symposium, 2005 IEEE*, pages 75–81. IEEE, 2005.

- [148] Yan Wu and Yiannis Demiris. Learning dynamical representations of tools for tool-use recognition. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2664–2669. IEEE, 2011.
- [149] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*, pages 5285–5294, 2017.
- [150] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
- [151] Jaewook Yoo, Jaerock Kwon, and Yoonsuck Choe. Predictable internal brain dynamics in eeg and its relation to conscious states. *Frontiers in Neurorobotics*, 8:18, 2014.
- [152] Keyan Zahedi, Nihat Ay, and Ralf Der. Higher coordination with less control a result of information maximization in the sensorimotor loop. *Adaptive Behavior*, 18(3-4):338–355, 2010.