Using Reinforcement Learning to model Collective and Adversarial Behaviours with Active Brownian Particles

Tham Yik Foong and Tom Oakes
School of Physics and Astronomy, University of Nottingham.
(Dated: August 2020)

Collective motion including flocking and clustering can often emerge from a system consisting of many active matters. For instance, flocks of bird, micro-organisms or even crowd of peoples. External force can be applied on such active matters, for example, Active Brownian Particles to perform collective behaviour, and such force can be approximated via Reinforcement Learning. We proposed an approach that combines model-based learning with temporal different learning that allowed us to train particles species that learn to perform flocking and clustering motion, utilizing the external force. We also demonstrated how by placing both species of particles in the same system can form adversarial behaviour against other species, which resemble the prey-predator model.

I. INTRODUCTION

The field of active Brownian particles is a well-studied field that has provided many toy models used to understand complex biological systems such as bird flocking and bacterial behaviour [1–3]. One such toy model is a system of active Brownian particles that are given individual directions that can be independently acted on by external forces. The forces acting on these particles can be described by particle-particle repulsion and noise, active Brownian particles, for example, can include a force governed by the environment or neighbouring particles.

The force used to cause complex behaviour such as flocking can be thought of as a functional approximation that causes the alignment to occur. Through Machine Learning (ML), specifically Reinforcement Learning (RL), the functional approximation can be discovered to produce a multitude of behaviours, and the system as a whole can be studied to see how the functional approximations affect various behaviours.

With these principles, it is possible to model different species of particles that can perform different behaviour. For example, species that prioritize flock or clustering. If successful, we can observe the dynamics of such particles in a system. On top of that, we can simulate different species of particles in the same system, and such a system can be projected as the predator-prey model.

Many recent works have demonstrated training active matters to flock via Reinforcement Learning [4–7]. The environment from these works tends to be simpler or grid-like. From a physical perspective, it is worth posting the question of, will such active Brownian particles, or in the language of RL, *agents* work under a more complex model? Specifically, in [4], agents have learned to perform collective motion in a simpler environment, we aim to observe what extend will the approach used in [4] can perform when migrating to a more complex environmental model.

Additionally, each of the learned behaviours can be driven by different intentions. For instance, in [5–8], agents learn to flock to avoid predators. Furthermore, shown in [4], agents learn to perform flocking to keep group cohesion and minimize neighbours lost. Nevertheless, it is worth studying how such behaviour can emerge from other intentions. It is suggested in [9] that collective motion and dynamic arrest of Brownian particles can emerge by sampling from large deviations. Such large deviations can come from the active work for repulsive active Brownian disks. Therefore, active work can be a good candidate in driving the agents to learn to perform different behaviours.

Besides that, it is possible of seeing how multiple species of agents can interact with each other in the same system. Both flocking and clustering behaviour agents alone are considered as collective behaviour agents, where agents will try to create spontaneous emergence of ordered movement. Putting them together in a same system creates a model that both species of agents, become adversarial against each other, will compete to optimize their own interest, resembling the prey-predator model. Related works show the possibility of creating such a learnable prey-predator model under the framework of Multi-agent Reinforcement Learning [10, 11].

Understanding such behaviour dynamics can bring insight into flocking or swarm behaviour of many species emerge in nature [3, 12–14]. Another important field of application is swarm robotics or control of nanomotors [15, 16], where different autonomous agents could perform different manoeuvre patterns according to its objective and environment. This work could also be used to model prey flocking in the presence of a predator where the learned behaviour of the predator species and prey species are different.

The objective of this work is to train agents that can perform flocking behaviour and clustering behaviour via RL. Once learned, observe the dynamics of placing multiple types of agents within the same environment. Also with the condition of agents trained under an environment that is more physically realistic.

This paper is structured as follows. In Section II, we give an overview of the model used to simulate our environment and required preliminaries on the subject of RL. In Section III, we defined the Markov Decision Process (MDP) adopt by our agents and show our approach

on training the agents. Followed by that, in Section IV, we describe the setup used training and evaluating the learning agents and the result of each training session. In Section V, Justification is provided on the result shown in Section IV. Finally, we conclude our work and discuss the potential future work as an extension of this paper.

II. BACKGROUND

A. Model

We replicated our model, or term that is more relevant to RL, *environment* from [9]. The model consists of N Brownian particles moving in a two-dimensional space. The space itself is governed by periodic boundary conditions (PBC), where it wraps around at the edges. For instance, particles that leave the square visualization on the right will immediately reappear on the left. Particles move around at a constant speed, and it can interact and collide with each other via an interaction force F_i. The movement mechanism of particles can be described as below:

$$r_i = \mu F_i + v_p u(\theta_i) + \sqrt{2D\eta_i} \tag{1}$$

and

$$\theta_i = E_a + \sqrt{2D_r}\xi_i \tag{2}$$

Where r_i and θ_i are the dimensionless position and orientation of particle respectively. They evolved over updates performed at discrete time steps dt with Eqn. 1 and 2. η_i, ξ_i are zero-mean unit-variance Gaussian white noises, representing the noise created from random and rapid collisions due to density fluctuations in space, such as fluid. μ indicates particle's mobility, v_p its bare self-propulsion speed, $u(\theta_i) = (cos\theta_i, sin\theta_i)$ its orientation vector, and D and D_r are translational and rotational diffusivities. we set $D_r = v_p/l_p$, where $v_p = 1$ is the self-propulsion speed and $l_p = 60$ is the persistence length. translational diffusivities is derived as $D = \sigma_p^2 D_r/3$, σ_p is the radius of particle. E_a represent the external force acting on the particles, such force allow agents to manipulate particles' orientation. When agents do not act on the particles, orientation of particle is simply just $\theta_i = \sqrt{2D_r}\xi_i$.

The interaction force F_i extends from the dimensionless WCA potential [17, 18]

$$F_i = [4((1/d)^{12} - (1/d)^6) + 1]\Theta(2^{1/6} - d) \tag{3}$$

where Θ is the Heaviside step function and d is the distance to other particles at position r_j. Keep in mind that, when computing distance d among particles, PBC need to be taken into account in the case where particles move in a limitless wrap space.

All environmental parameters used in this simulation are summarized in Table I. Except for N and dt, which values are modified depends on the occasion, other environmental parameters remain the same throughout this thesis.

TABLE I. Environmental parameters, notation and value used in simulation.

Parameters	Notation	Value
Persistence length	l_p	60
Particle's mobility	μ	1
Self-propulsion speed	v_p	1
Particle's radius	σ	1
Rotation diffusion	D_r	v_p/l_p
Translation diffusion	D	$\sigma_p^2 \times D_r/3$

B. Reinforcement Learning

The study of Reinforcement Learning (RL) focus on building an agent that learns while interacting with its environment [19]. The mere objective for that agent is to learn a policy and maximize its cumulative reward. Agent often involved in an iterative process of evaluating its behaviour, or *policy* $\pi(s)$ in the jargon of RL, and improve on top of it.

The interaction of agent and environment is formalized as Markov Decision Process (MDP) [10, 19], where environment provides the agents with any state from state space $s \in S$, and the agents will act based on it by providing an action from action space $a \in A$. In return, the environment provides the reward r based on reward function R and the next state s' based on agents' action. f is the state transition probability function, which fully (only when the environment is deterministic) describe the dynamics of MPD $f := p(s', r|s, a)$. MDP can be formalized as a tuple $\langle S, A, f, R\rangle$.

Utilizing the MDP formalization, an agent can find the value of a particular state subjected to some policy function $\pi(s)$ using value function. State value function is defined by

$$V_\pi(s) = E[R|s, \pi] \tag{4}$$

Thus, value of state s can be iteratively update using

$$V(s) \leftarrow V(s) + \alpha[R - V(s)] \tag{5}$$

where α is a step-size parameter, which influences the rate of learning. Other than state value, one can also estimate state-action value by

$$Q_\pi(s, a) = E[R|s, a, \pi] \tag{6}$$

It means finding the value of performing a certain action a in a given state s following policy π. The Eqn. 6 can be iteratively updated using

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R - Q(s,a)] \tag{7}$$

This is used as an update rule in [4], and also as the initial baseline method we used to train our agent. Last update rule that will be mentioned in this section is called a temporal-difference learning method (TD-Learning), defined as

$$V(s) \leftarrow V(s) + \alpha[R + \gamma V(s^{'}) - V(s)] \tag{8}$$

where changes of state value are based on a difference between estimates of future state and current state $V(s^{'}) - V(s)$, hence so-called temporal-difference (TD). γ is a discount factor that tells how important future rewards are to the current state.

Finding an optimal policy π^* usually involve a process called policy iteration, which consists of two steps: policy evaluation and policy improvement. Value-based method mentioned above can be considered as policy evaluation. With the estimation of state value and policy evaluation, we can further improve on agents policy using policy improvement.

The temporal credit assignment problem [19, 20] is a common issue often discussed in the RL domain which also appeared in this experiment when using approach from [4]. It refers to the problem when reward is terribly temporally delayed and the immediate action will not receive a relevant reward distant in the future, this happens especially when the time steps are very fine-grained, such as in our case.

C. Multi-Agent Reinforcement Learning

When an RL system involves multiple agents, the problem is elevated to another paradigm known as Multi-Agent Reinforcement Learning (MARL). This subject presents multiple agents that share the same environment to be either self-interested (maximizing individual reward) or working together to achieve a cooperative goal (maximizing global reward) [10, 11, 21].

Training agents that work toward a common goal is known as full cooperative. When a system involves agents that have opposing goal, the interaction is competitive among different species but cooperative among the same species [10, 11, 22]. In this thesis, it is interpreted as a cooperative setting when training collective behaviour agents that are homogeneous. While training adversarial behaviour agents that are heterogeneous is determined as a competitive setting

The generalization of MDP in multi-agent case is a tuple $\langle S_1 \cdots S_N, A_1 \cdots A_N, f, R_1 \cdots R_N \rangle$ where N is the number of agents. $S_i, i = 1 \cdots N$ is the state of agent i, when agents gain the full state of the environment, for example, all agents share the same vision, it can be defined as a full state. Otherwise, it is often denoted as $o_i \in O$ for observation, where o_i is the individual observation of an agent since other agents' action is not visible to the agent. However, in this thesis, we treat the action of other agents as part of the environment S_i for each agent i. We denote the joint state of all agent as S.

$A_i, i = 1 \cdots N$ is the available actions return from agents (actions can be produced from a shared policy or joint policy), yielding joint action set A. f is the state transition probability function. When the environment solely returns a global reward, it can be denoted as a scalar R_G. global reward R_G is then shared across all agents. However, when each agent is assigned with distinct reward, $R_i, i = 1 \cdots N$ is the reward to agents and can be joint as a reward set R.

New challenges, such as the reward credit assignment problem [23–25] arise, where proper feedback needs to be assigned to the agent responsible for creating good or bad reward signals. Another challenge is the problem of non-stationary environment [10, 26], where one agent need to adapt its actions in certain states, while other agents (which can be considered as the environment to the first agent) are constantly changing theirs as well. Therefore, agents need to account for other agents' behaviour. Multiple agents learning concurrently can often cause RL methods failed to converge to an equilibrium state. There exist many other challenges in the MARL domain, however, only these two problems are encountered in our experiment.

III. METHOD

This section explains the learning technique used by our agents.

A. Markov Decision Process

State Space This experiment comprises of two different types of agents: flocking behaviour agent Ag_f and clustering behaviour agent Ag_c. Both agents have a different set of state space. The state of flocking behaviour agents s_i are modelled as the angular difference between the agents and the average orientation of neighbouring agents ϑ_i [4].

The orientation of neighbouring agents ϑ_i can be described as

$$\vartheta_i = (\textstyle\sum_{|r_j - r_i| < Rg} \theta_j)/n_i \tag{9}$$

n_i is the number of neighbouring agents within a range $Rg = 3$, and

$$s_i = ad(\vartheta_i, \theta_i) \tag{10}$$

where function $ad(\vartheta_i, \theta_i)$ is defined as $arccos(\vartheta_i \cdot \theta_i / \|\vartheta_i\|)$ for $\vartheta_i \cdot (\theta_i)_\perp > 0$ with $(\theta_i)_\perp$ obtained by rotating $\pi/2$ counter clockwise the unit vector θ_i, and minus this quantity otherwise. This indicates that the angular difference with respect to agents' current orientation is $s \in [-\pi, \pi)$. To formulated our approach as tabular solution method, state s is divided into K pieces of descritized unit. We set $K = 20$, thus, number of state is $K_s = 20$ [4]. See FIG. 1.A for visualization of state space of flocking behaviour agents.

On the other hand, clustering behaviour agent's state space is modelled as the angular difference between the agent and the average position of neighbouring agents within range Rg:

$$\phi_i = (\textstyle\sum_{|r_j - r_i| < Rg} r_j)/n_i \tag{11}$$

and

$$s_i = ad(\phi_i, \theta_i) \tag{12}$$

Refer to FIG. 1.B for visualization of state space of clustering behaviour agents.

When putting flocking behaviour and clustering behaviour agents in the same system, both type of agents no longer consider clustering behaviours agents in range as neighbour. Instead, only flocking behaviour agents are considered as neighbour. In other words, when computing ϑ_i or ϕ_i, r_j and n_i will not take clustering behaviour agents into account (see FIG. 1.C and FIG. 1.D).

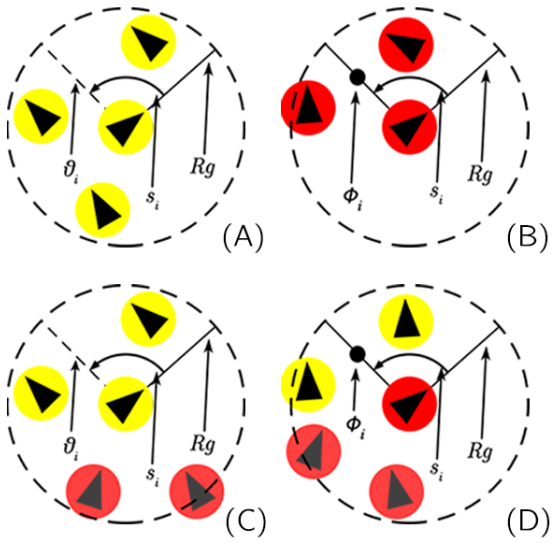

FIG. 1. Visualization of components in each type of agents state spaces. (A) Flocking behaviour agents (yellow cells) perceive average orientation ϑ_i (dashed line) of neighbouring agents within Rg (dashed line circle), and state $s_i \in [-\pi, \pi)$ is the angular difference of agents orientation θ_i with ϑ_i. (B) Clustering behaviour agents (red cells) first compute the average position ϕ_i (point) of all neighbouring agents within Rg, and find its state $s_i \in [-\pi, \pi)$ by computing the angular difference of agents orientation θ_i with ϕ_i. In the adversarial case, (C) flocking behaviour agents and (D) clustering behaviour agents disregard neighbouring clustering behaviour agents (semi-transparent red cells) as its neighbour when computing ϑ_i and ϕ_i.

Action Space The action of agents a is just the external force E_a act on the agents' orientation. The action space of agents includes only the discretized angle of which it desires to turn at each time step.

The velocity of agents always remain constant and can not be altered throughout the experiment. Keep in mind that, angle is discretized by $K = 20$ pieces, thus, indicate that there are $K_a = 20$ possible turning angles, equally spaced in the agents action space $a \in [-\pi, \pi)$.

Rewards The reward function is initially modelled as a cost of losing neighbouring agents within its perception range Rg [4], defined as

$$R_i = \begin{cases} n_i^{t_1} - n_i^t, & \text{if } n_i^{t+1} < n_i^t \\ 0, & \text{if } otherwise \end{cases} \tag{13}$$

where n_i^t is the number of current neighbours and $n_i^{t+1} - n_i^t$ is the amount of neighbours lost during the time step. Therefore, the goal of our learning agents is to avoid losing neighbouring agents that can lead to punishment. This reward function is later replaced by another alternative reward function.

In [9], it is shown that collective motion and dynamic arrest of Brownian particles can be optimized and measured using active work. Active work measures how far the local self-forcing of particles translates into real motion. Current active work of an individual agent is defined as $w_i = r_i \cdot u(\theta_i)$, and the current total active work done by all the agents is $W = \frac{v_p}{\mu} \sum_{i=0}^{N} w_i$. The normalized current active work per agent is $\varpi = W/N$. The reward function is thus described as below:

$$R_i = \Omega(\varpi + w_i) \tag{14}$$

where Ω is a scalable hyperparameter that help controls the prioritization over active work. For negative Ω, agents will prioritize achieving high active work, otherwise if Ω is positive.

Agents can be trained using only global active work as reward, even so, it is tested that including individual active work can help to boost convergence speed and performance [23]. The intuition behind this is individual active work aid to create another learning target for our agents, since maximizing individual active work also help to maximize global active work.

B. Q-Function

A Q-value table is constructed to store the state-action value, it is a matrix with size $(K_s \times K_a)$. The Q-value is updated via

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha[R_i - Q(s_i, a_i)] \quad (15)$$

which is Eqn. 7 mentioned in Section II B. Q-value is iteratively updated by sampling from experiences produced by the environment.

ϵ-greedy policy is used as an approach to ensure agents get enough exploration regarding the environment. It helps agents to discover more rewarding action set, and avoid being trapped in a sub-optimal policy. with ϵ-greedy policy, agents will select random actions from a uniform distribution with ϵ probability. Otherwise, with $1 - \epsilon$ probability, the agent will always select the action that yield the highest value in a given state $argmax_a Q_i(s, a)$ (also refers as greedy action).

Such learning rule is adopted by [4] and used as our initial approach for training agents value function. Note that this approach of estimating value function is later replaced by the TD-Learning in Section III C.

C. Model-Based Learning with TD-Learning

Policy iteration can be split into two components: policy evaluation and policy improvement. To evaluate value that agent might receive while being on a certain state with a fixed policy, a temporal difference learning rule defined by

$$V(s_i) \leftarrow V(s_i) + \alpha[R_i + \gamma V(s_i^{'}) - V(s_i)] \quad (16)$$

is used to perform policy evaluation. Learning rate is denoted as α, which determines the weight when updating toward a target state value. γ is a discount factor multiplying (decaying) the future expected reward. The term $\gamma V(s_i^{'})$ is an estimate for the true value $V(s_i)$, and is often referred as *bootstrapping* in reinforcement learning when learning involved one or more estimated values in the update step for the same kind of estimated value.

On the other hand, Policy is controlled by a table lookup distribution model, where stochastic state transition function $P(s^{'}|s, a)$ are learned to perform planning and action selection. Every interaction between agents and environment will produce experience $(s, a, r, s^{'})$, and a state transition $(s, a, s^{'})$ is used to update a counter $n(s, a, s^{'})$. Every occurrence of state transition $(s, a, s^{'})$ increase counter by $n(s, a, s^{'}) \leftarrow n(s, a, s^{'}) + 1$. State transition function can then be updated via $P(s^{'}|s, a) \leftarrow \frac{n(s,a,s^{'})}{n(s,a)}$. Since the environment dynamic is stochastic, it is possible to contain several next states and next rewards, each with some probability of occurring.

Experience is sampled from the environment each time agents interact with it. All sampled experience is stored within an experience buffer. Learning is performed when the buffer size exceeds a certain threshold of β. This prevents instability of learning by providing a fixed target value for the estimated value function to approximate with [27].

ϵ-greedy policy is used. Therefore, when selecting a greedy action, the agent will always select the action with the highest probability that can lead to the estimated optimum state, based on state value estimated. It is possible that optimum state can not be reached with any action, in this case, look for the next optimum state until any action can be taken. Such action selection function can be described as:

$$a(s) = \begin{cases} \text{random action}, & \text{with prob. } \epsilon \\ argmax_a(P(s^*|s, a)), & \text{with prob. } 1 - \epsilon \end{cases} \quad (17)$$

where s^* being the local optimum state an agent can reach with its available actions.

Method that required a model of the environment is known as model-based reinforcement learning [19, 28, 29]. Model of the environment often store estimates of the MDP's dynamics $P(s^{'}, r|s, a)$, and can be used to generate simulated experience without interacting with the environment itself or can be used for planning (for example Dyna-Q [19, 30]). However, the estimated reward is not stored in our approach. This is because of the delayed reward and the perturbed reward can incur inaccuracy of the model itself. Therefore, it is replaced by an estimated state value function.

Each homogeneous agent shares the same policy but updates it independently, the rate of updating the policy and model is multiplied by the number of agents per time step, thus making the same policy converge much quicker [22].

D. Teacher-learner model [4]

N_T number of teachers are introduced in the system along with our agents. These teachers contain a hard-wired policy based on modified version of the Vicsek model [12], allowing them to flock by default. Teachers first obtain the average orientation of its neighbouring teacher by $\vartheta_i = (\sum_{|r_j - r_i| < R} \theta_j)/n_i$, then angular difference, σ_i of θ_i and ϑ_i is computed. The orientation of teacher is then updated with $\theta_i \leftarrow \theta_i + \sigma_i$

Thus, the teachers move in a robust flocking pattern. In contrast, the agents, which is the learners, does not have a fixed policy and must be trained over time to acquire a valid policy. The number of teachers N_T and the number of learners, denoted as N_L, create a total of N agents within the same space.

E. Regularization

Epsilon Decay In order to balance exploration and exploitation behaviour, epsilon ϵ is set to decay exponentially over time. It makes sense for the agent to explore more initially, and exploit more as it approaches its target policy, or in a sense, understand its environment more. The decay of ϵ is controlled by:

$$\epsilon \leftarrow \epsilon_{min} + (\epsilon_{max} + \epsilon_{min})e^{-\lambda_\epsilon episode} \tag{18}$$

where ϵ_{min} and ϵ_{max} is the minimum and maximum of ϵ. *episode* is the current episode and λ_ϵ being the decay rate within range [0, 1]. Higher decay rate implies that agent exploits its greedy action sooner during training, and it is possible that agent might not be able to find an optimal policy before ϵ decay to 0, especially for task with sparse reward. On the other hand, low λ_ϵ value required more training time but is more likely that agent can find its optimal policy.

Learning Rate Decay Learning rate decay is a technique often implemented in modern machine learning project and is empirically observed to help both the optimization and generalization during training [31]. Applying learning rate decay prevents the agents from "unlearning" its solution. Decay of learning rate α followed a similar formula from Eqn. 18:

$$\alpha \leftarrow \alpha_{min} + (\alpha_{max} + \alpha_{min})e^{-\lambda_\alpha episode} \tag{19}$$

where α_{min} and α_{max} is the minimum and maximum of α respectively. λ_α is the decay rate for α within range [0, 1]. A large λ_α would result in a learning rate that is too small for the agents to learn effectively.

With all the approach introduced in this section, the workflow of training collective behaviour agents can be summarized in Algorithm 1. We follow the design framework of *OpenAi Gym* [21] with regard to the interaction between environment and agent. Since policy and model are shared across multiple agents, only one agent class is required.

The only difference in training adversarial behaviour agent compare to a collective behaviour agent is declaring an extra number of agent class. State value function and model is not shared across different species of agents. Some hyperparameters, however, can be shared across different agent type, for example, ϵ and α. The workflow of training adversarial behaviour agent can be summarized as Algorithm 2.

Algorithm 1 Training Collective Behaviour Agents

Initialize value function $V(s)$ and model $P(s^{'}|s,a)$ arbitrary
Initialize learning rate α, discount rate γ, and exploration rate ϵ
Initialize learning rate decay λ_α, and exploration rate decay λ_ϵ
Set buffer size β
while current episode $\leq$ total episode **do**
    Initialize Environment
    Initialize total time step τ
    Initialize current time step t with 0
    $S \leftarrow$ return initial joint state from environment
    while t $\leq \tau$ **do**
        Sample $\omega \in (0, 1]$ from a uniform distribution
        if $\epsilon > \omega$ **then**
            Take action A from agents
        else
            Take action that can lead to the optimum state given current state $A \leftarrow argmax_a(P(s^*|s,a))$, if optimum state can not be reach with any action, look for the next optimum state until any action can be taken
        end if
        Observe $S^{'}, R$ from environment
        Store experience tuple $(S, A, S^{'}, R)$ in experience buffer
        Update counts $n(s,a,s^{'}) \leftarrow n(s,a,s^{'}) + 1$
        Compute $n(s,a) \leftarrow \sum_{k=1}^{K_s} n(s,a,s_k^{'})$
        Update agent model $P(s^{'}|s,a) \leftarrow \frac{n(s,a,s^{'})}{n(s,a)}$
        if *experience buffer size* $> \beta$ **then**
            $V(s) \leftarrow V(s) + \alpha[R + \gamma V(s^{'}) - V(s)]$
            Empty experience buffer
        end if
        Increase t by time step dt
    end while
    $\epsilon \leftarrow \epsilon_{min} + (\epsilon_{max} + \epsilon_{min})e^{-\lambda_\epsilon (episode)}$
    $\alpha \leftarrow \alpha_{min} + (\alpha_{max} + \alpha_{min})e^{-\lambda_\alpha (episode)}$
end while

IV. RESULT

This section describes the setup used for configuring and evaluating the learning agents under various environment setting.

A few important environmental parameters need to be carefully considered. For instance, the density of the two-dimensional space that contains the moving agents can influence the active work of the system itself. A population density that is too high can induce lower active work as agents are compressed together in a compact space, and agents can hardly move in this case. In contrast, Low density can cause agents failed to extract any useful information during training. This is due to that agents can rarely collide with each other in the same space and the system will always remain in high active work, hence, performing any action is less likely going to affect the system active work and the reward returned.

Other configurable environmental parameters followed the standard value given in Table I. Agents are trained with a time step *dt* of 10^{-3} (0.01 when performing trial-and-error with different variant of approaches and 1 when testing interaction force is absent) and total time step τ of 100 iterating over 120 episodes. Each episode starts by

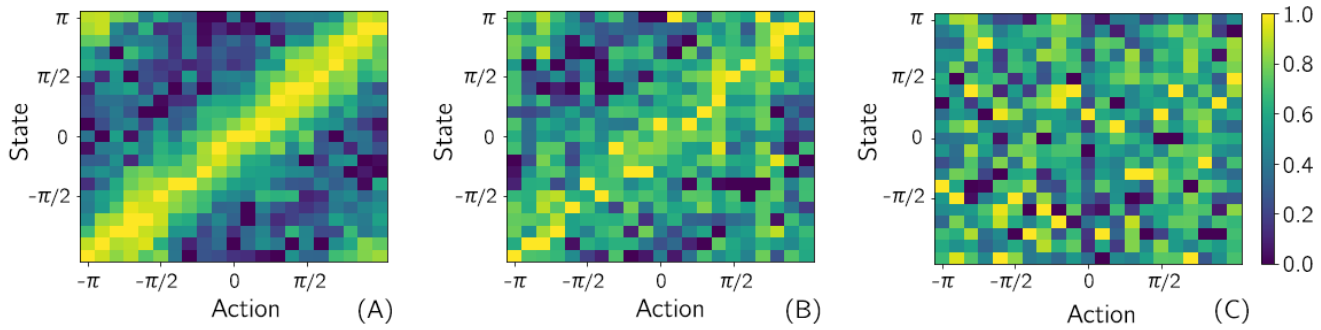

FIG. 2. Q-table of agents training using Q-function. Colour represents the normalized state-action value, with yellow being closer to 1 and navy closer to 0. When selecting a greedy-action, state-action value with the brightest yellow will be selected. (A) Q-table of agent trained in the environment without interaction force and noise. (B) Q-table of agent trained in the environment with interaction force but without noise. (C) Q-table of agent trained in the environment with interaction force and noise, the complex environment we adapt for the rest of the experiments.

Algorithm 2 Training Adversarial Behaviour Agents

Initialize agents Ag_f *and* Ag_c
Set buffer size β
while current episode $\leq$ total episode **do**
  Initialize Environment
  Initialize total time step τ
  Initialize current time step t with 0
  $s_f, s_c \in S \leftarrow$ *return initial joint state from environment and split into two state sets* s_f, s_c *for* Ag_f *and* Ag_c
  while t $\leq \tau$ **do**
    return action a_f *from* Ag_f *based on* s_f
    return action a_c *from* Ag_c *based on* s_c
    joint a_f *and* a_c *to form joint action A and return A to environment*
    Observe $S^{'}, R$ *from environment*
    Store experience tuple $(S, A, S^{'}, R)$ *in experience buffer*
    Update $P_{Ag_f}(s^{'}|s,a)$ *and* $P_{Ag_c}(s^{'}|s,a)$
    if *experience buffer size* $> \beta$ **then**
      $V_{Ag_f}(s) \leftarrow V_{Ag_f}(s) + \alpha[R + \gamma V_{Ag_f}(s^{'}) - V_{Ag_f}(s)]$
      $V_{Ag_c}(s) \leftarrow V_{Ag_c}(s) + \alpha[R + \gamma V_{Ag_c}(s^{'}) - V_{Ag_c}(s)]$
      Empty experience buffer
    end if
    Increase t by time step dt
  end while
  $\epsilon \leftarrow \epsilon_{min} + (\epsilon_{max} + \epsilon_{min}) e^{-\lambda_\epsilon (episode)}$
  $\alpha \leftarrow \alpha_{min} + (\alpha_{max} + \alpha_{min}) e^{-\lambda_\alpha (episode)}$
end while

instantiating N number of agents in the space with random initial position and orientation. Every new episode starts with a transient phase, where agents are moving randomly to ensure the randomness of each episode. After the transient phase, agents begin to learn and act accordingly to the state.

Episode reached its termination state when $t = \tau$. When an episode is terminated, the total active work done per agent [9] throughout the training session, defined as

$$aw(t) = \frac{v_p}{\mu N} \int_0^t dt \sum_{i=0}^{N} w_i \tag{20}$$

and $aw(t = \tau)$, is recorded for evaluation, and the environment will be reset for the next episode. Experiment with the same configuration is run multiple times to obtain the median performance result, increasing confidence in calculating an accurate measurement of performance.

A. Flocking Behaviour

We begin by training flocking behaviour agents. Before implementing the full model from Section II A, we first experiment with a simpler version of the particles movement mechanism. Experiment is performed on each iteration of adding new environment complexity, such as interaction force and noise. Performance is then evaluated with each iteration of evolvement by inspection on simulation and the Q-value table. This approach provides an insight into how the environment complexity is going to affect agents' performance, and if the approach is suitable for the training agents in such an environment.

We implement the learning rule from Section III B as an initial approach. State space of flocking behaviour agent used Eqn. 10. We used lost of neighbouring agents as our reward function (Eqn. 13). Additionally, teacher-learner model is also implemented, we set the number of teachers N_T and the number of learners $N_L = 50$, making up to total $N = 100$ agents in the same space.

Again, before implementing the complex behaviour of the Brownian particles from [9], we first simulate a simpler version of it, where position and orientation of agents can be described as $r_i = v_p u(\theta_i)$ and $\theta_i = E_a$. This removed the interaction force and movement noise in Eqn. 1 and 2, and reproduce the movement mechanism used

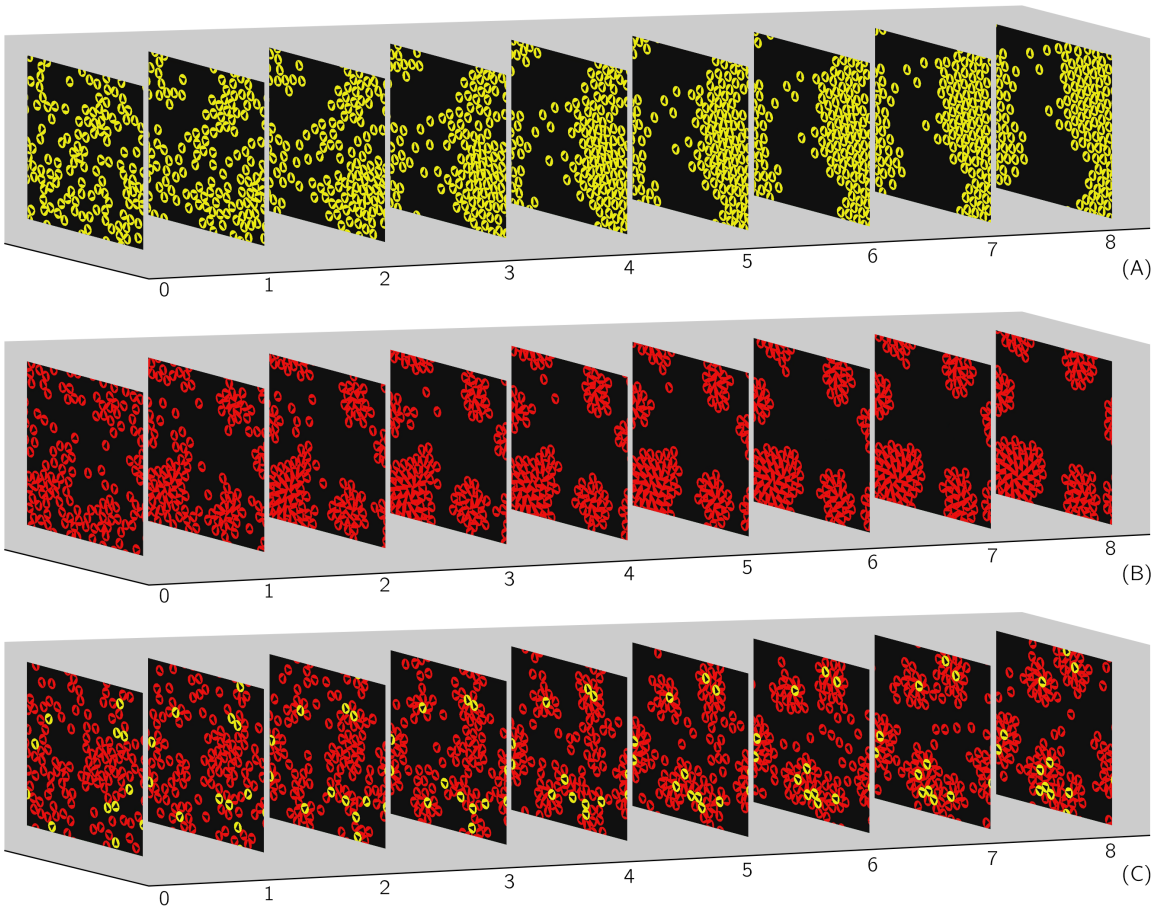

FIG. 3. Above plot shows transition of collective and adversarial behaviour agents within $t = 8$ time step (x-axis). Agents position and orientation are randomly initialized and only starting to act after a transient phase. (A) Flocking behaviour agents (yellow cells) forming alignment and perform flocking. (B) Clustering behaviour agents (red cells) forming several clusters. (C) Clustering behaviour agents (red cells) forming cluster around flocking behaviour agents (yellow cells).

in [4]. Time step dt for this training session is set to 1 to accommodate the setting of [4]. FIG. 2.A shows the Q-table after training the agents for 120 episodes, which resemble the result shown in [4]. This proves that the result of [4] is replicable using a similar environmental setting. Agents trained under such condition is able to perform flocking along with teachers.

We further evolve the model to a more complex setting, which is adding the interaction force between agents. We modified the position update equation to $r_i = \mu F_i + v_p u(\theta_i) + \sqrt{2D\eta_i}$. Now that agents can collide with each other, a smaller dt is required, because a larger dt can potentially cause agents to repel from each other distance away when getting too close, affecting the accuracy of environment. Thus, dt is set to 0.01 when training agents that have interaction force included. FIG. 2.B shows the Q-table of the trained agent. Although it is observed that the Q-table still forms a similar pattern, some greedy actions are incorrectly mapped.

Moreover, η_i and ξ_i are added to the position and orientation update equation respectively (Eqn. 1 and 2) as zero-mean unit-variance Gaussian white noises. At this point, our model has implemented the Brownian particles model from Section II A. FIG. 2.C shows the trained agents' Q-table. It is apparent that agents failed to extract a meaningful feature from the environment, instead, the mapping of state-action value look chaos. The trained agents in this case only perform random motion and did not show any particular motion pattern.

We discover that this approach is not suited for the newly adopted environment with noise and interaction force, and therefore switched to the approach mentioned in Section III C to perform the remaining experiments.

We use model-based learning approach to train our agents with the complex model. For the following experiments, we no longer use loss of neighbours as cost, instead, we use the normalized current active work per agent ϖ as our new global reward function $R_G = \Omega\varpi$. The reward function is computed with $\Omega = -1$ to encourage maximization of system active work.

Different variants of approaches are tested to evaluate the efficiency of each variant. These include the usage of learning rate decay, aggregating global reward with individual reward, and bootstrapping. We first train

agents that do not implement learning rate decay, only use global reward and use learning rule mentioned in Eqn. 5 without bootstrapping. Result of this method can be seen in FIG. 4, the average active work of the last 10 episodes is 0.87. Despite this basic approach manages to train agents that are capable of performing flocking motion, the end result is worst than other methods with enhancement. We proceed to improve on top of this method by adding new regularization methods.

We add learning rate decay (see Eqn. 19) as regularization method and train our agents to compare the result. we set the learning rate decay $\lambda_\alpha = 0.2$. Looking at plot in FIG. 4, learning rate decay help to increase the performance by 4%, with an average of active work of last 10 episodes equal to 0.9221. However, the convergence rate remains the same, where agents only start to perform at its peak after 40 episodes.

That being said, we reshape our reward function by adding individual reward with global reward (Eqn. 14). This creates another learning target for our agents. We train our agents and yield a result showing in FIG. 4. It is notable that not only convergence speed has increased drastically, the performance has improved as well. Agents are able to converge to its optimum policy after 20 episodes. Average active work of the last 10 episodes is now 0.971.

The last modification on our base method is changing the update rule (Eqn. 5) to TD-Learning (Eqn. 16) by adding bootstrapping. Now that our agents have implemented full method introduced in Section III C and III E, including bootstrapping, global reward and individual reward, and learning rate decay. The final result of this approach is shown in FIG. 4. The increase of convergence speed is notable, and performance remains almost identical, which the average active work of the last 10 episodes is 0.973.

Combination of these methods discussed is proven to be effective in our simulation. Hence, are used in our subsequent experiment on training collective behaviour agents and adversarial behaviour agents.

Up to this point, experiment on flocking behaviour agents still has the teacher-learner model applied. We move on to the situation where there is no teacher in the environment. Thus, $N_T = 0$ and $N_L = 100$ to keep the same density in space. The training yield an average active work of the last 10 episodes of 0.972. The comparison in FIG. 5 shows the result of training with and without teacher. We find that even without teacher, agents is able to learn flocking to maximize active work. This gives us a reason to drop the teacher-learner model in future experiment.

Previously, we trained our agent using time step $dt = 0.01$ to increase the computational speed to search for practical approach while also maintain the environment accuracy. Now that we have concluded the approach that can train our agents effectively, we further decrease dt to 10^{-3} to train our agent more accurately. FIG. 6.A shows the progression of active work when training flocking behaviour agents throughout 120 episodes. The average active work of the last 10 episodes is 0.9572. Fig. 3.A demonstrates how trained flocking agents start to form alignment and flock.

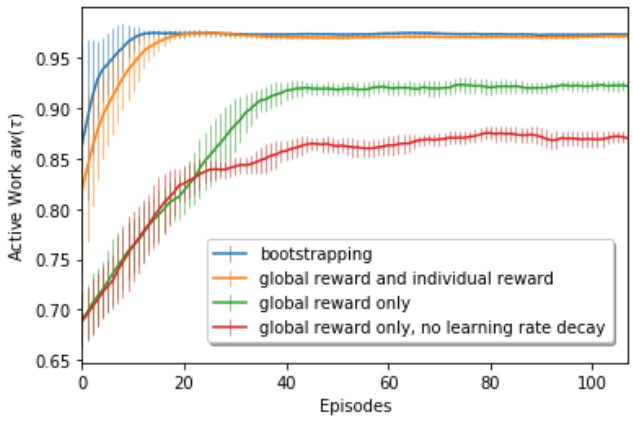

FIG. 4. Above plot shows the active work across 120 episodes. Curves are smoothed with a moving average over 13 points. We compare our method with four different variants of methods.

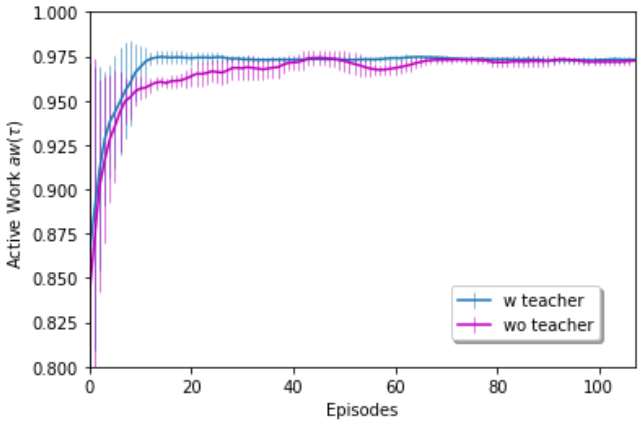

FIG. 5. The comparison of agents training with and without teachers. Lines are smoothed with a moving average over 13 points. Agents trained with teachers (blue line) contain $N_T = 50$ teachers and $N_L = 50$ learners. Agents trained without teachers (magenta line) contain only $N = 100$ agents or learners.

FIG. 6.A (inset) also shows how the heat map of state value function progress during training. Flocking behaviour agents will prioritize aligning its orientation with its neighbouring agents. Agents will turn toward the average orientation of neighbouring agents in range when not align. Otherwise, agents will remain its orientation.

Agent hyperparameters is summarized in Table II.

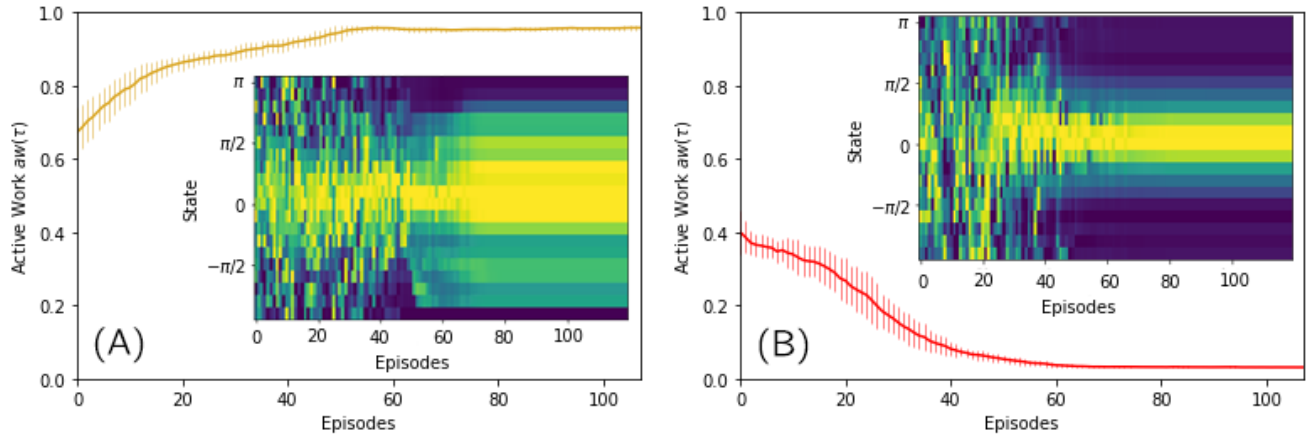

FIG. 6. Above figures show how active work generated by (A) flocking and (B) clustering behaviour agents progress throughout 120 episodes of training. Curves are smoothed with a moving average over 13 points. Insets show the state value function progress over the training session, colour represents the normalized state value, with yellow being closer to one and navy blue the opposite.

TABLE II. Agents hyperparameters, notation and value used to generate result in FIG. 6.A.

Parameters	Notation	Value
Number of agents	N	100
Range	Rg	3
Active work prioritization	Ω	-1
Time step	dt	10^{-3}
Total time	τ	100
Total episode		120
Buffer size	β	10^4
Learning rate	α	0.1
Learning rate decay	λ_α	0.2
Minimum learning rate	α_{min}	0
Maximum learning rate	α_{max}	0.1
Exploration rate	ϵ	1
Exploration rate decay	λ_ϵ	0.1
Minimum exploration rate	ϵ_{min}	0
Maximum exploration rate	ϵ_{max}	1
Discount factor	γ	0.8

B. Clustering Behaviour

The second type of agents we trained is the clustering behaviour agents. The objective of agents, in this case, is trying to suppress system active work as much as possible. We inherit the approach used to train flocking behaviour agents, however, there are a few differences on how parameters are configured. Contradict to flocking behaviour agent, the reward function is computed with $\Omega = 1$ alternatively to minimize the active work. Furthermore, this type of agent used state space mentioned in Eqn. 12.

The outcome of this training is a type of agents that learned to perform clustering motion to reduce active work (see FIG. 3.B), which lead to dynamical arrest. It shows that agents will try to form a large or several small clusters which can suppress their movement. When agent's perception range Rg is high, it is more likely all agents in the same space will form a large cluster. Otherwise, several small clusters are formed. State progression map in FIG. 6.B (inset) indicates that agents can gradually learn that facing toward the average position of neighbouring agents can receive a higher return, hence will try to prioritize being in such state. The graph in FIG. 6.B also shows that agents can maintain active work as low as 0.0311 toward the end of the training.

This proves that by simply changing the state space and set Ω to 1, agents can learn with the same approach shared by another type of agents and achieve a different set of objective.

TABLE III. Agents hyperparameters, notation and value used to generate result FIG. 6.B. Most hyperparameters shared the same value with Table II, below table only shows the values differ from it.

Parameters	Notation	Value
Active work prioritization	Ω	1

C. Adversarial Behaviour

So far we have only been training cooperative homogeneous agents in the same scenario, moving forward, we extend our model by placing the trained flock behaviour agents and clustering behaviour agents together in the

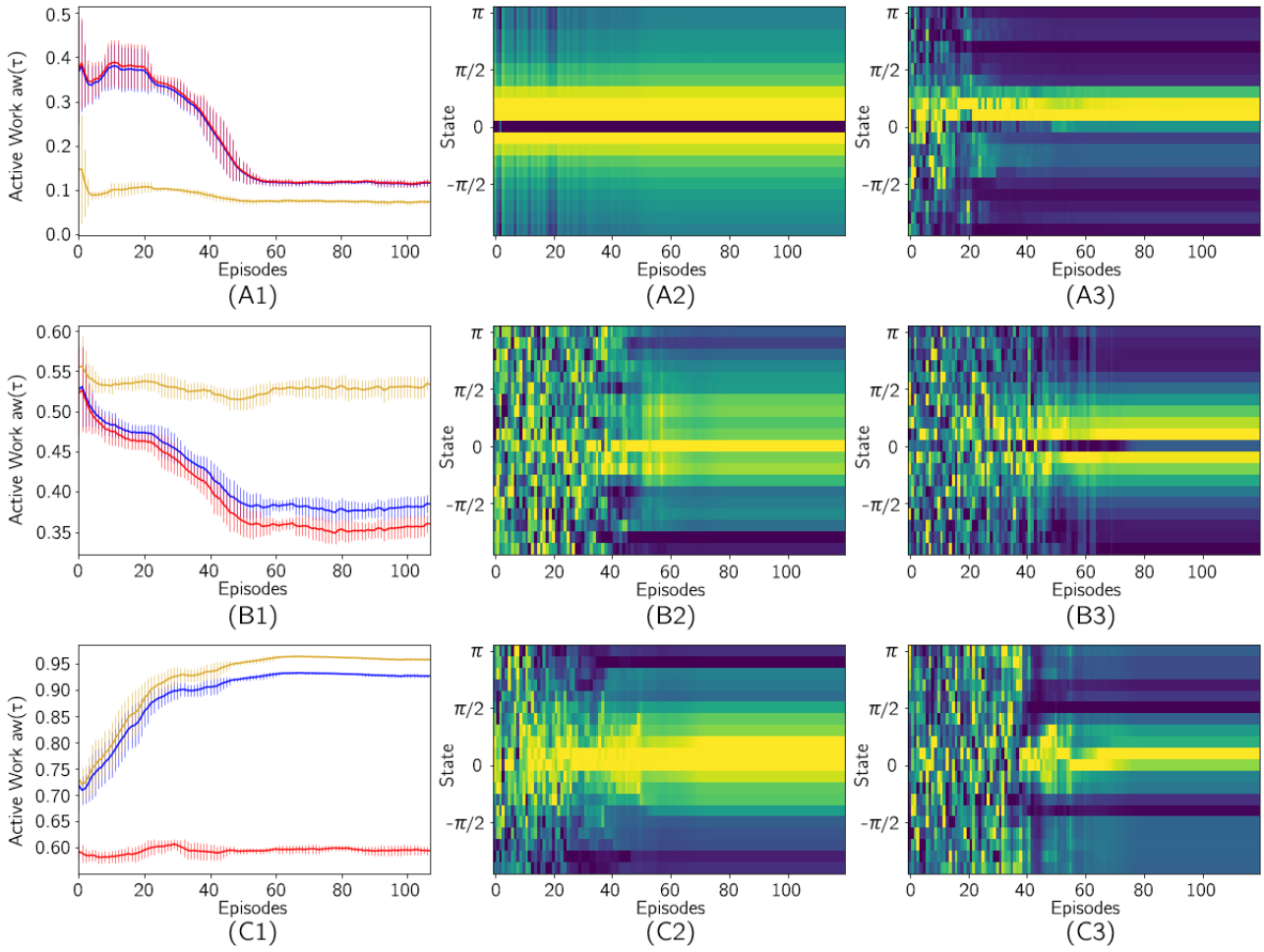

FIG. 7. Above plots are generated by training adversarial behaviour of flocking and clustering behaviour agents in the same system throughout 120 episodes with different population ratio. rows A, B, C are results that trained under the ratio of $\frac{1}{34}$, $\frac{1}{6}$, $\frac{32}{3}$ of flocking behaviour agents respectively. (A1, B1, C1) show the active work for all agents (blue line), flocking behaviour agents only (yellow line) and clustering behaviour agents only (red line), the lines is smoothed by a moving average of 13 points. Heat maps (A2, B2, C2) show the progression of state value for flocking behaviour agents throughout 120 episodes, and (A3, B3, C3) for clustering behaviour agents. Colour represents the normalized state value, with yellow being closer to 1 and navy closer to 0.

same system. This form a competitive heterogeneous agents environment, where both types of agents become adversarial against the other and aim to either maximize the active work or to reduce it. Moreover, this also leads to a situation called *zero-sum game*, in which each type of agents' gain or loss of reward is exactly balanced by the losses or gains of the utility of the other type of agents [10]. In other words, the only way to increase reward is to exploit the benefits of other types of agents.

When training both agents in the same system, their policy and transition model is transferred to a new training session, without having to retrain again from ground up. Unlike previously, ϵ is set to 0.2 at the start of each training session.

We run the training with a population ratio of Ag_f to Ag_c being 1 to 6. Indicated in [9], agents under this configuration required six others agents to be completed surrounded (see FIG. 9.B). Hence, we select this ratio to avoid unfair competition. The outcome of the training is two types of agents that behave like FIG. 3.B. Clustering behaviour agents learned to form a cluster that surrounds flocking behaviour agents. flocking behaviour agents, on the other hand, remain flocking which can potentially prevent clustering behaviour agents' capture or even break up a formed cluster surrounding an agent, by distracting them.

FIG. 7.B1 shows the active work for all agents, flocking behaviour agents only and clustering behaviour agents only. The average active work of the last 10 episodes is summarized in Table IV.

From Section IV A and IV B, we know that in a simulation that contains only cooperative homogeneous agents, both types of agents can either reach an active work as high as 0.9572 or as low as 0.0311. In a competitive

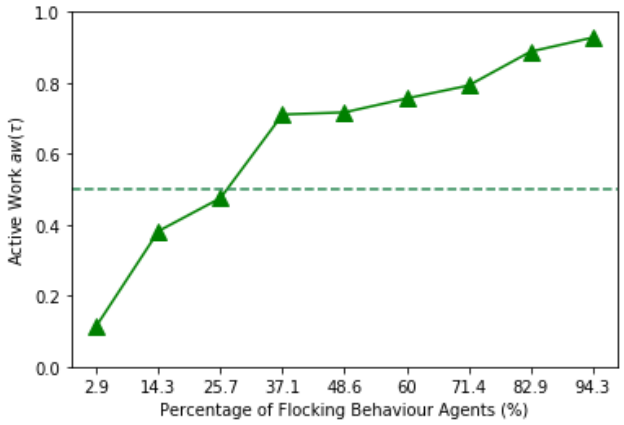

FIG. 8. Active works generated from different population percentage of flocking behaviour agents in percentage. population N is set to be 35, composed from N_f flocking behaviour agents and N_c clustering behaviour agents. N_f and N_c varies depend on the percentage. Dashed line is drawn at the point of $aw(\tau) = 0.5$

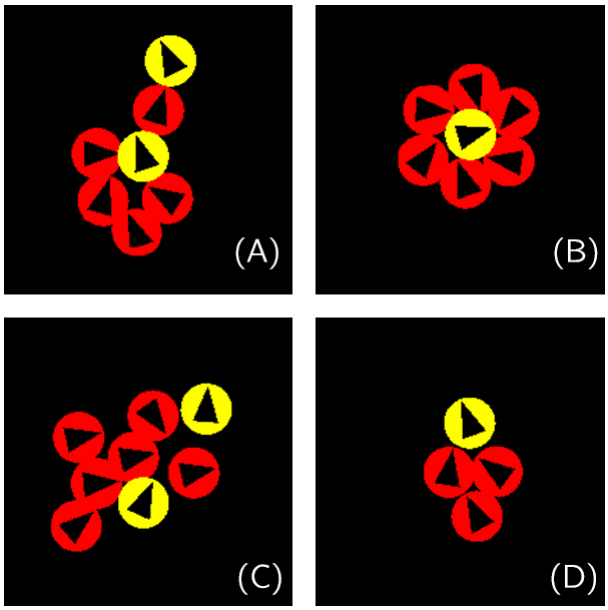

FIG. 9. Notable behaviours occur among our trained adversarial behaviour agents, or prey-predator model. (A) Prey (yellow cell on top) "saves" another captured prey (yellow cell at the centre) by distracting predators (red cells). (B) It requires a minimum of six predators to completely surround a prey, to suppress its movement. (C) Predators distracted by multiple preys. (D) Predators chasing after a prey.

setting, however, global active work starts from 0.5043 and gradually decrease until it reached an equilibrium state at 0.3828 after 50 episodes. The equilibrium state is known as *Nash Equilibrium* in Multi-agent system language, meaning that both agents have obtained the equilibrium strategies for its opponents, and the strategies will remain the same until one party changes their policy.

TABLE IV. The active work of flocking behaviour agents, clustering behaviour agents and all agents trained in the same system. The active work is averaged over the last 10 episodes of the training session.

Type of agents	Active work
All agents	0.3822
Flocking behaviour agents	0.5310
Clustering behaviour agents	0.3574

Clustering behaviour agents slowly adopt a new policy to reduce active work, which can be reflected on its state value progression heat map (see FIG. 7.B3). Agents no longer prioritize pointing straight toward the average position of neighbouring agents. Instead, agents learned to point slightly toward the sides. Conversely, the active work of flocking behaviour agents remain in a steady state throughout the whole training session. Its state value function progress the same way as in cooperative setting (see FIG. 7.B2). This means that the best option is still perform flocking, even when clustering behaviour agents will try to form a cluster around it to reduce its velocity.

TABLE V. Agents hyperparameters, notation and value used to generate result FIG. 7. Most hyperparameters inherit its value from agents hyperparameters that trained flocking and clustering behaviour agents, below table only shows the values differ from it.

Parameters	Notation	Value
Number of flocking behaviour agents	N_f	Depends on ratio
Number of clustering behaviour agents	N_c	Depends on ratio
Range	Rg	2
Active work prioritization of flocking behaviour agent	Ω_{Ag_f}	-1
Active work prioritization of clustering behaviour agent	Ω_{Ag_c}	1
Learning rate	α	0.1
Exploration rate	ϵ	0.2

We proceed to experiment with both species of agents with different population ratio, this guides us to figure how many more population of agents need to exceed to show dominance over the other species of agents. We reduce N to 35 to increase computational speed. 9 training sessions are executed with different percentage of flocking behaviour agents, FIG. 8 shows the outcome of this experiment. As the percentage grows, active work increases and enters a steady state with high active work as it reaches a point when flocking behaviour agents occupy approximately 40% of the system.

Other than that, we plot the progression of active

work, progression of state value of flocking behaviour agents and clustering agents for ratio $\frac{1}{34}$, $\frac{1}{6}$, $\frac{32}{3}$ (see FIG. 7). When there exists only one single flocking behaviour agent, active work can remain as low as 0.1152 averaged by the last 10 episodes. Clustering behaviour agents will behave by pointing straight toward flocking behaviour agents (FIG. 7.A3). Progression of state value of flocking behaviour agent can be ignored. Since no neighbour agents can be perceived by it as $N_f = 1$, and ϑ will always remain 0, hence, learning can not occur.

When the amount of clustering behaviour agents is overwhelmed by flocking behaviour agents, the active work of the system remains as high as 0.9276 (FIG. 7.C1). Flocking behaviour agents inherit the same behaviour from previous collective motion training (FIG. 7.C2). Clustering behaviour agents learned to point straight toward flocking behaviour agents (see FIG. 7.C3).

V. DISCUSSION

We demonstrated that agents failed to learn a proper mapping of state-action value from a complex model with noisy scenarios and temporal delayed reward using Q-function. The main reason being that extreme temporal delayed reward exists when time step is very fine-grained, and the agents failed to map immediate state-action with the reward received. Moreover, when noise is involved, depending on the environment and reward function, it can potentially form a perturbed reward. Perturbed reward can cause agents to receive arbitrary errors or unlearning correct representation of state-action value.

We then proceed to implement model-based learning to train our flocking behaviour agents. Utilizing state value learned, we perform planning using a trained state transition probability function, this help to remove the need of learning a Q-table. Decoupling value of a state-action pair with the reward received immediately using state value function help to bypass the issue of temporal delayed reward mentioned above. Notably, agent can learn state value more accurately than state-action value with this environment dynamic.

By performing trial-and-error, we learned a combination of methods that can enhance our method more efficiently. Learning rate decay is generally applied when training a neural network [31]. Learning rate decay helps the agents converge to a local minimum and avoid oscillation. Agents might unlearn a correct policy as training progress, reducing learning rate helps agents to be more confidence toward its policy when approaching the end of the training session.

We discover that aggregating global reward with individual reward helps to boost convergence speed and performance to a great extend. Doing so help to assign feedback to the agent responsible for its action, overcoming the issue of credit assignment issue. In the case of training flocking behaviour agents, the overall system might generate high active work. However, individual agent might perform invalid action but still receive high global reward, and assigning low individual reward help to punish such individual. Besides, individual reward help to create another learning target other than just global reward, since maximizing individual reward essentially helps to maximize global reward in our model.

Bootstrapping is used by many reinforcement learning methods as it can help agents to learn faster, which implied by FIG. 4. Intuitively, to reach an optimum state, it is required to be in the state before it. Therefore, Propagation of estimated value of the successive state to its previous state help to fasten learning. However, it might not be clear how significantly bootstrapping has increase learning since the upper bound of the result is limited by agents' exploration rate at early stage of training.

We then test our model with and without the teacher-learner model. As expected, although agents are able to learn without teacher, the convergence speed drops slightly. This is also demonstrated in [4], but changes in convergence speed are not reported. Nevertheless, we decided to pull off teacher-learner model for the rest of experiment, as we could not find an accurate model that can represent the teacher for clustering behaviour agents (unlike teacher from flocking behaviour agents, which is based on a modified version of the Vicsek model).

The policy of trained flocking behaviour agents has enabled agents to flock. Agents will keep alignment and group cohesion with its neighbouring agents. This is reasonable, as to maximize active work, agents with constant propelling speed simply have to keep its velocity and produce efficient motion. However, colliding with other agents can result in decreased velocity. Collision happens more often when population density is high. Therefore, the ideal case to maximize active work is to align all agents and move toward the same direction with a constant speed, minimizing collision.

Many recent works [4–7] have also successfully simulated flocking motion using reinforcement learning. What differentiates our flocking behaviour agents with agents from these work is the intention that drives agents to perform flocking. Most agents perform flocking to avoid predators [5–7] or solely to maintain neighbours count [4]; Whereas our agents learn flocking to increase system active work.

In contrast, trained clustering behaviour agents will perform clustering. Agents will aggregate and form a large or several small clusters. Agents accumulated in a region of high population density, especially in the centre of a cluster, resulting in almost no velocity due to compression from other agents. Therefore, clustering can cause low active work which is the objective of clustering behaviour agents.

From FIG. 3.C, the observed dynamics between two species of agent somewhat correspond to prey-predator model from nature. Clustering behaviour agents, as a predator learned to chase after or capture a flocking behaviour agents by surrounding it. We notice that the final formation of learned state value (see FIG. 7.B3) proposed

that pointing directly toward flocking behaviour agents will yield lower reward. This is due to all agents share a constant propelling speed, agents can never catch up to another agent when both moving toward the same direction (see FIG. 9.D). Therefore, blindly chasing after another agent can induce high active work let alone not being able to capture it.

Flocking behaviour agents, as the prey in this case, learned to distract predator by flocking. We hypothesized that agents might try to distract predator by avoiding the same species by going another direction when one is in range. Surprisingly, collective behaviour agents still prefer to flock. After observing the simulation, we are aware that agents are more likely to be captured when left alone, and flocking can distract predator as predictor failed to focus on a single target position (see FIG. 9.C). Another interesting phenomenon is when a prey passes by another surrounded prey, it can distract the surrounding predators and "free" the captured prey (see FIG. 9.A).

Nonetheless, flocking behaviour agents failed to avoid incoming clustering behaviour agents, which is usually an important feature shown in the prey-predator model [5, 8]. This is due to the lack of a rich state space, where there is no feature showing the presence of prey. Adding another dimension on state space, such as average position of neighbouring prey, can potentially help with the issue. However, this can prompt the volume of state space to increase, requiring more computational resources.

As perception range of agents Rg increased, predator is able to find further target, but prone to distraction by other prey as well. This can affect the performance of agents when assigned with an incorrect value. Another trade-off needs to be carefully considered it population density, as high-density favour clustering behaviour agents (lower active work) and the opposite favour flocking behaviour agents (higher active work). As mentioned earlier, homogeneous agents shared the same policy and update on it independently, multiplying learning speed by the number of agents. This approach can lead to unfair competition when the number of agents is different among agent types. However, its negative effect remains unclear as none is observed.

We also experiment with how changing the percentage of flocking behaviour agents population can affect the system's active work. Note that due to constraint on time and computational resources, we could not reflect an accurate result by running more instances of training. Nevertheless, it is sensible that the active work grows as the population ratio of flocking behaviour agents increase, because of an individual flocking behaviour agent required six or more clustering behaviour agents to completed surrounded. In addition, more flocking behaviour agents also means increased distraction for clustering behaviour agents to maintain a cluster. We concluded that beyond the percentage of 25%, flocking behaviour agents starting to show dominance over clustering behaviour agents, if we use active work $aw(\tau) = 0.5$ as a baseline.

VI. CONCLUSION

In our thesis, we have demonstrated approaches attempted to train agents that perform collective motion and adversarial behaviour. We proved that Q-function failed to map state-action value within the context of fined-grained time unit and perturbed reward. We then tested out model-based learning along with other regularization tricks such as learning rate decay, aggregating global and individual reward, and TD-Learning.

We concluded that model-based learning with TD-Learning (with a few other learning tricks) can train agents effectively, overcoming the issue of complex environment and temporal delayed reward. Using MARL framework, we trained two types of agents that learned to either maximize or minimize active work. Flocking behaviour agents maximize their active work by performing flocking motion, which helps to keep group cohesion and suppress collision between agents. Whereas clustering behaviour agents learned to form dense clusters and reducing agents motion to minimize the active work.

We transferred the previously trained agents and place them into a same system, doing so allow us to acquire a prey-predator model to investigate with. Clustering behaviour agents learned a policy that can capture flocking behaviour agents without blindly chasing it. While flocking behaviour agents' strategy is to perform flocking and distract clustering behaviour agents. Last but not least, we study the population percentage needed for one party to gain dominance over the other.

It is mentioned that in our model of prey-predator, prey failed to avoid incoming predator due to lack of feature within agents state space. It is worthwhile to attempt enhancing agent's sensory. It can be done by embedding predator's position information into state space, or even model agent's vision as a vector that store visual information returned from casting rays around the body of particles.

Doing so also lead to a future direction of work, which is learning via function approximation, such as neural network. With a function approximation model, it is easier to adapt to high dimensional state space. Finally, it is possible to add obstacles and objects into the environment, which can create a richer dynamics for collective motion model or prey-predator model.

Code and media files can be found in this Github repository: `https://github.com/zfoong/collective-x-adversarial-ams`

[1] T. Vicsek and A. Zafeiris, Collective motion, Physics Reports **517**, 71–140 (2012).

[2] A. Cavagna, A. Cimarelli, I. Giardina, G. Parisi, R. Santagati, F. Stefanini, and M. Viale, Scale-free correlations in starling flocks, Proceedings of the National Academy of Sciences **107**, 11865 (2010), https://www.pnas.org/content/107/26/11865.full.pdf.

[3] K. Drescher, K. C. Leptos, I. Tuval, T. Ishikawa, T. J. Pedley, and R. E. Goldstein, Dancingvolvox: Hydrodynamic bound states of swimming algae, Physical Review Letters **102**, 10.1103/physrevlett.102.168101 (2009).

[4] M. Durve, F. Peruani, and A. Celani, Learning to flock through reinforcement, Physical Review E **102**, 10.1103/physreve.102.012601 (2020).

[5] C. Hahn, T. Phan, T. Gabor, L. Belzner, and C. Linnhoff-Popien, Emergent escape-based flocking behavior using multi-agent reinforcement learning (2019), arXiv:1905.04077 [cs.MA].

[6] P. Sunehag, G. Lever, S. Liu, J. Merel, N. Heess, J. Leibo, E. Hughes, T. Eccles, and T. Graepel, Reinforcement learning agents acquire flocking and symbiotic behaviour in simulated ecosystems (2019) pp. 103–110.

[7] H. La, R. Lim, and W. Sheng, Hybrid system of reinforcement learning and flocking control in multi-robot domain, (2020).

[8] J. Schrum, *Competition Between Reinforcement Learning Methods in a Predator-Prey Grid World*, Tech. Rep. AI08-9 (The University of Texas at Austin, Department of Computer Sciences, 2008).

[9] T. Nemoto, Fodor, M. E. Cates, R. L. Jack, and J. Tailleur, Optimizing active work: Dynamical phase transitions, collective motion, and jamming, Physical Review E **99**, 10.1103/physreve.99.022605 (2019).

[10] K. Zhang, Z. Yang, and T. Başar, Multi-agent reinforcement learning: A selective overview of theories and algorithms (2019), arXiv:1911.10635 [cs.LG].

[11] L. Busoniu, R. Babuska, and B. De Schutter, Multi-agent reinforcement learning: An overview (2010) pp. 183–221.

[12] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, Novel type of phase transition in a system of self-driven particles, Physical Review Letters **75**, 1226–1229 (1995).

[13] J. Buhl, D. J. T. Sumpter, I. D. Couzin, J. J. Hale, E. Despland, E. R. Miller, and S. J. Simpson, From disorder to order in marching locusts, Science **312**, 1402 (2006).

[14] I. Buttinoni, J. Bialké, F. Kümmel, H. Löwen, C. Bechinger, and T. Speck, Dynamical clustering and phase separation in suspensions of self-propelled colloidal particles, Phys. Rev. Lett. **110**, 238301 (2013).

[15] A. Altemose and A. Sen, Chapter 11: Collective behaviour of artificial microswimmers in response to environmental conditions (2019) pp. 250–283.

[16] M. Ibele, T. Mallouk, and A. Sen, Schooling behavior of light-powered autonomous micromotors in water, Angewandte Chemie (International ed. in English) **48**, 3308 (2009).

[17] D. M. Heyes and H. Okumura, Some physical properties of the weeks–chandler–andersen fluid, Molecular Simulation **32**, 45 (2006), https://doi.org/10.1080/08927020500529442.

[18] G. S. Redner, M. F. Hagan, and A. Baskaran, Structure and dynamics of a phase-separating active colloidal fluid, Phys. Rev. Lett. **110**, 055701 (2013).

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction, second edition* (MIT Press, 2018).

[20] M. Minsky, Steps toward artificial intelligence, Proceedings of the IRE **49**, 8 (1961).

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, Openai gym (2016), arXiv:1606.01540 [cs.LG].

[22] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in *In Proceedings of the Tenth International Conference on Machine Learning* (Morgan Kaufmann, 1993) pp. 330–337.

[23] D. T. Nguyen, A. Kumar, and H. C. Lau, Credit assignment for collective multiagent rl with global rewards, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18 (Curran Associates Inc., Red Hook, NY, USA, 2018) p. 8113–8124.

[24] Y. han Chang, T. Ho, and L. P. Kaelbling, All learning is local: Multi-agent learning in global reward games, in *Advances in Neural Information Processing Systems 16*, edited by S. Thrun, L. K. Saul, and B. Schölkopf (MIT Press, 2004) pp. 807–814.

[25] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, Counterfactual multi-agent policy gradients (2017), arXiv:1705.08926 [cs.AI].

[26] S. Padakandla, P. K. J., and S. Bhatnagar, Reinforcement learning algorithm for non-stationary environments, Applied Intelligence 10.1007/s10489-020-01758-5 (2020).

[27] H. van Hasselt, A. Guez, and D. Silver, Deep reinforcement learning with double q-learning (2015), arXiv:1509.06461 [cs.LG].

[28] L. Kuvayev and R. S. Sutton, Model-based reinforcement learning with an approximate, learned model, in *in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems* (1996) pp. 101–105.

[29] M. Kearns and S. Singh, Near-optimal reinforcement learning in polynomial time, Mach. Learn. **49**, 209–232 (2002).

[30] R. S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, SIGART Bull. **2**, 160–163 (1991).

[31] K. You, M. Long, J. Wang, and M. I. Jordan, How does learning rate decay help modern neural networks? (2019), arXiv:1908.01878 [cs.LG].