2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Hierarchical Reinforcement Learning for Non-Stationary Environments

1st Rachel Haighton
Department of Systems and Computer
Carleton University
Ottawa, Canada
rachelhaighton@cmail.carleton.ca

2nd Amirhossein Asgharnia
Department of Systems and Computer
Carleton University
Ottawa, Canada
amirhosseinasgharnia@sce.carleton.ca

3rd Howard Schwartz
Department of Systems and Computer
Carleton University
Ottawa, Canada
schwartz@sce.carleton.ca

4th Sidney Givigi
School of Computing
Queen's University
Kingston, Canada
sidney.givigi@queensu.ca

***Abstract*—What indications are there when the environment changes and the learned policy is no longer optimal? Is it possible to predict when a non-stationary environment changes in some way? In this paper we propose a method that helps agents know when to retrain their policies via reinforcement learning. The agents detect changes based on the temporal difference. A hierarchical learning model is used to aid in these non-stationary environments. The hierarchical learning model has two levels, the higher-level policy, which we call the learning switch, and the lower-level policy, which tells the agents their suitable action to play the game. The higher-level policy determines when reinforcement learning should be turned on or off based on the temporal differences calculated within a game or episode. Two multi-agent differential games are used as examples. The first two examples tackle the problem in cooperative games, while the last example addresses the competitive game. The results show that the agents can maintain suitable performance by switching on the learning process for a few iterations after environment changes occurs. In this paper we consider the change in environment to be long term occurrence within the dynamics of games; for example the mass of an agent may become heavier.**

***Index Terms*—Fuzzy Systems, Multi-Agent Reinforcement Learning, Non-stationary Environment, Learning Rate**

I. Introduction

Learning new parameters in a controller network can be computationally expensive, which is why it is important to stop learning once a model has been adequately trained [1]. However, if the model's environment has changed how do we know when to turn learning back on and update the network parameters to reflect the new environment? An environment where changes occur repeatedly is often noted as a Non-Stationary Environment [2]. In this paper, the dynamics of the environment will change abruptly and stay that way until another change occurs. The environmental changes are changes in the dynamics of the game, for example the mass of the agent may increase abruptly. This paper addresses a supervisory reinforcement learning (RL) structure that turns the learning on, when it is necessary, and turns the learning off, when the parameters converge.

To learn more efficiently in a non-stationary environment, we propose a hierarchical learning model. This solution is independent of the game being played in a decentralized multi-agent system as it uses the temporal difference (TD) to determine environment changes. The hierarchical learning model has two levels, the higher-level policy (HLP), which we call the learning switch, and the lower-level policy (LLP), which returns the primitive action to the agents to play the game. The HLP is to determine when reinforcement learning should be turned on or off based on the lower-level TD calculated within a game or episode. The TD is often called the prediction error and is used to update the controller network. Here we use it to determine if the update is necessary by inputting it's statistics into the higher-level network in the hierachical learning model.

In this paper, we propose the idea of a hierarchical learning switch in the context of a competitive and a cooperative decentralized multi-agent reinforcement learning game. When the environment has changed, the HLP sends a signal to turn learning on in the LLP. Once the network parameters have successfully adapted to the new environment, the HLP sends a signal to turn the LLP learning off. This is advantageous as computational time may be reduced by allowing the learning to automatically be turned off.

We use the fuzzy actor-critic learning (FACL) algorithm in both the higher and lower levels of the hierarchical structure. Fuzzy inference systems (FIS) are used as the actor and critic. A major strength of fuzzy reinforcement learning is adapting the fuzzy system parameter weights when the environment changes. Since each fuzzy rule has a meaning with respect to the inputs, the values can adapt to these changes accordingly without needing to start learning a new policy.

The contribution of this paper is twofold: 1) we proposed a finite horizon model-free method to turn the learning on and off based on the environment situation through the use of temporal differences; 2) we examined the proposed method using two multi-agent games: one cooperative and one competitive

game.

This paper is organized in six sections. In section II, we present some of the most recent papers on reinforcement learning. In section III, we present the FACL algorithm and the games we used as the simulation platform. Section IV is dedicated to the proposed method. In section V, we present our results and discussion. Finally, section VI finalizes the paper by the conclusion and references.

II. Literature Review

The idea of using RL in differential games is not new and there is a plethora of papers on the subject. Over 20,000 pieces of literature have been published addressing reinforcement learning in non-stationary environments between 2015 to 2023 alone. We address a few articles, where non-stationary environments are most relevant to the present work.

The idea of using temporal differences as a learning indicator was proposed in [3]. As the temporal differences tend toward 0, it becomes more obvious that the adaptation is converging to the correct values.

In [4], deception is studied in pursuit-evasion games. Deception was modeled using a hierarchical policy scheme. The lower-level policy was trained using the FACL algorithm, while a genetic algorithm was used to train the higher-level policy. Although the hierarchy shows a promising performance in increasing the evader's outcome, the proposed method suffers from lacking of robustness against the changes in the environment, such as changing the initial location of the agents. Some parts of the problems coping with a non-stationary environment were solved in [5], where the FACL algorithm was used to train the higher-level policy as well.

Ideally one way to deal with changes in environment is to have all the variables as the decision-maker's input. However, the authors in [4], [5] limited the number of inputs for the trained controllers, to deal with the curse of dimensionality. In addition, in both papers, the learning rates decay over time, which narrows the learning window in the final learning iterations.

The authors in [6] proposed a framework for studying deception in the context of optimal control. It considers a scenario where an agent has an objective to achieve, but there is an adversary who aims to learn the agent's intentions and prevent it from succeeding. In this adversarial setting, the agent has an incentive to deceive the adversary while working towards its objective. A deceptive environment where an adversary changes its strategy can be regarded as a non-stationary environment. In [6], the problem was formulated as convex and non-convex optimization problems.

To tackle the problem, an object-oriented method was proposed in [3], where the learning rates were not decaying during the game. Therefore, the algorithm was able to continue learning efficiently, even if a change happens in the final iterations. The game in [3] is a combination of cooperative and competitive games: a multi-agent pursuit-evasion game is modeled by introducing one evader and three pursuers.

In [7], the authors leverage the success of deep reinforcement learning and utilize a deep Q-Network (DQN) to encode observations of opponents. Instead of explicitly predicting the opponent's actions, the opponents' observations are incorporated into the DQN. This allows the agent to learn and adapt its own policy based on the observed behaviors of opponents.

In [8], the Switching Agent Model (SAM) is a proposed approach to address the challenges of learning in non-stationary environments. It combines deep reinforcement learning with opponent modeling and utilizes uncertainty estimations to switch between multiple policies. This framework aims to improve the performance of traditional deep reinforcement learning methods, which often struggle in such dynamic settings. In [8], different policies are learnt beforehand and in the required situation they are implemented.

In [9], the authors investigate representation learning for decision-making in multi-task scenarios with changing environments. The authors propose an online algorithm that learns and transfers representations, referred to as a low-dimensional feature extractor, for tasks within each environment. These representations differ across environments. The algorithm aims to enhance decision-making efficiency in non-stationary environments. The paper provides theoretical analysis and experimental results using synthetic and real data to demonstrate the algorithm's superior performance compared to existing methods that treat tasks independently.

Ref. [10] focuses on reducing energy consumption in HVAC systems while maintaining occupant comfort. It introduces a novel approach called non-stationary DQN that combines active detection of changes in the building environment with deep Q network (DQN) for HVAC control. The method aims to address the challenge of non-stationarity by identifying environmental changes and learning effective control strategies accordingly. Simulation results demonstrate that the proposed method outperforms existing techniques, achieving significant energy savings of up to 13% and improving thermal comfort by 9% in both single-zone and multi-zone control tasks. The method also exhibits stability in the presence of disturbances and demonstrates the ability to generalize well to new building environments, highlighting its robustness and potential for practical implementation.

In all the mentioned literature, there is no mechanism to switch learning on and off based on the game states. In [1], the authors propose a switching mechanism that turns the learning on and off. The work introduces a model-free RL method called Context Q-learning, designed to learn optimal policies in dynamic environments. It incorporates a new change detection algorithm to identify changes in the environment models alongside Q-learning. The policies learned by the method perform well in varying operating conditions and provide a higher return compared to the classical Q-learning.

III. Fuzzy Actor-Critic Reinforcement Learning And The Games

This section describes the FACL algorithm, which is used as the learning algorithm for both the HLP and LLP. we also

describe the cooperative and competitive games used to test the proposed method. FACL is used for its interpretability and simplicity.

A. Fuzzy Actor-Critic Learning

The FACL algorithm is used for training the LLP and HLP. The algorithm was initially proposed in [11], as a tool to map a continuous input set to a continuous action. The actor stores the actions in each state, while the critic gives an estimation of future performance. A Takagi-Sugeno (TS) fuzzy logic controller (FLC) is utilized for the actor. The critic is a TS fuzzy inference system (FIS) and estimates the state values. The output of the actor is (1) where ω_t^l is the actor's output parameter of rule l, L is the total number of rules, and ϕ^l is the firing strength of the rule l.

$$u_t = \sum_{l=1}^{L} \phi^l \omega_t^l. \tag{1}$$

During training, noise is added to the output to mimic exploration. The noise is from a normal distribution with a mean of 0, and a standard deviation σ, noted as $n(0, \sigma)$. We refer σ as the exploration-exploitation factor, and is largely based on the game being played. The actor's output during the learning process is,

$$u_t' = u_t + n(0, \sigma). \tag{2}$$

The firing strength of the rule is represented by (3), and is shown as follows,

$$\phi^l = \frac{\partial u}{\partial \omega^l} = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i)}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i))}, \tag{3}$$

where $\mu^{F_i^l}(\bar{x}_i)$ calculates the membership degree of the input $\bar{x}_i$ with n being the number of inputs.

The actor gets updated every iteration by (4),

$$\omega_{t+1}^l = \omega_t^l + \beta \Delta_t \phi_t^l (u_t' - u_t), \tag{4}$$

where β is the learning rate of the actor, and Δ_t is the temporal difference at time t, and R_{t+1} is the reward at a given time step. The actor learning rate should be smaller than the critic learning rate to prevent instabilities in the actor. The term $(u_t' - u_t)$ is equivalent to the noise that was added to the system for learning and exploratory purposes.

The critic's task is to estimate the state value in each time step. After each action output by the actor, the critic evaluates the new state to check performance. The value functions at t and $t+1$ must be calculated in order to eventually update the output parameters of the fuzzy rules. The value function is simply the expected sum of discounted rewards and is approximated by the fuzzy inference system as:

$$V_t = \sum_{l=1}^{L} \phi_t^l \zeta_t^l \tag{5}$$

$$V_{t+1} = \sum_{l=1}^{L} \phi_{t+1}^l \zeta_t^l \tag{6}$$

Using the value function, we can estimate the prediction error or temporal difference (TD) as,

$$\Delta_t = r_{t+1} + \gamma V_{t+1} - V_t. \tag{7}$$

The discount factor, γ, is between 0 and 1. The discount factor can help control the time horizon of the agent and thus its priority of short-term rewards, additionally it helps with the stability of learning algorithms. The term r_{t+1} is the reward received which is based on the game. The critic output parameters in the fuzzy inference system can then be updated using the temporal difference at t and learning rate, α,

$$\zeta_{t+1}^l = \zeta_t^l + \alpha \Delta_t \phi_t^l. \tag{8}$$

B. Cooperative Game: Continuous Hallway

In this game, two agents are randomly placed in a continuous space hallway. They must make it to the end of the hallway at the same time. The hallway is 15 m long, and the agents must input a force, F, into the dynamics of the system:

$$m\ddot{x} + b\dot{x} = F, \tag{9}$$

where b is a coefficient of friction and m is the mass. The mass and coefficient of friction are changed abruptly in order to create the non stationary environment. The shaping reward is as follows,

$$\begin{aligned} r_{t+1} &= w(D_{ig}(t) - D_{ig}(t+1)) \\ &\quad + 0.01(1-w)\text{exp}(-(\frac{D_{ij}(t)}{0.1})^2), \end{aligned} \tag{10}$$

where w represents a weight. Here we use a weight of 0.98. The weight determines which part of the function to value more, in this case getting to the finish line is valued more than staying near the partner agent. In terms of the terminal reward, if one agent gets to end of the hallway alone, the agent receives a terminal reward of +3, whereas the other agent receives 0. If both agents get to the end at the same time, the agents both receives +15.

The inputs to the actor are the agent's distance to the end of the hall, the agent's velocity, the distance between the agent and the partner agent, and the partner agent's velocity.

C. Competitive Game: Guarding A Territory

Pursuit-evasion games have been studied since the 1950s to address differential games. Differential games are the generalized version of discrete games in game theoretical analysis, where they are played in the continuous time domain. Differential games are described via differential equations. In a pristine PE game, an evader strives to escape from a pursuer. The game finishes when the pursuer catches the evader. However, a more challenging class of PE games is called guarding a territory, where the invader (similar to the evader)

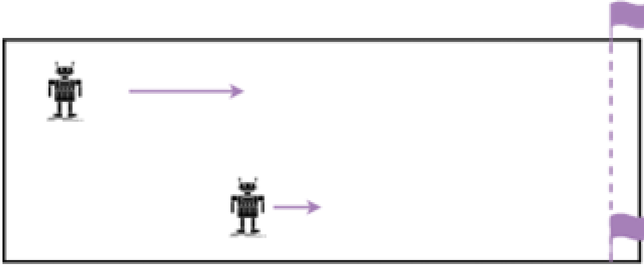

Fig. 1. Illustration of the Hallway Game

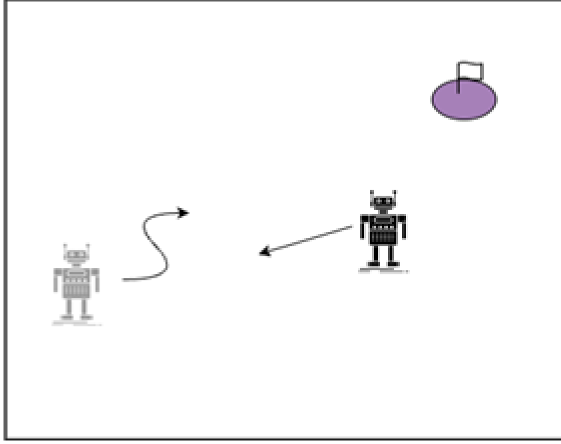

Fig. 2. Guarding A Territory Game Illustration

strives to get to a target, while escaping from a defender (similar to the pursuer). The defender tries to intercept the invader before the invader reaches the target. In this paper, we used the same autonomous robot model for both the invader and the defender. In a two-dimensional world, the differential equation for the invader and the defender is as follows,

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v\tan(\varphi)}{L} \end{cases} \tag{11}$$

where (x, y) is the Cartesian coordinate of the agent, θ is the agent's heading, φ is the robot's steering angle, L is the distance between the rear and the front axles, and finally v is the agent's speed.

Unlike the game in section III-B, this game is called a competitive game, since the invader's and the defender's desires are conflicting. The invader wins if it gets to the target and thus, the defender loses. The defender wins if it catches the invader, which is a failure for the invader. The competition is reflected in the reward functions. The invader's reward function is the same as [4] for the game is as follows,

$$\begin{aligned} R_{inv} = W_I(d_{IG}(t) - d_{IG}(t+1)) \\ + (1 - W_I)(d_{ID}(t+1) - d_{ID}(t)), \end{aligned} \tag{12}$$

where, R_{inv} is the invader's reward function, W_I is the invader's reward weight, $d_{IG}(t)$ is the distance between the invader and the defender at time step t, and Δt is the time step. On the other hand, the defender's reward function is as follows,

$$\begin{aligned} R_{def} = W_D(d_{DI}(t) - d_{DI}(t+1)) \\ + (1 - W_D)(d_{DG}(t) - d_{DG}(t+1)), \end{aligned} \tag{13}$$

where, R_{def} is the defender's reward function, W_D is the reward weight, d_{DG} is the distance between the invader and the defender at time step t.

Eqs. (12) and (13) are designed in a way that always return a signal in the interval of [0,1]. In [4] a method is proposed to set the reward weights W_I and W_D.

The invader's inputs are as follows,

$$\text{Invader's Inputs} = [\alpha_I \quad d_{IG} \quad \beta_I \quad d_{ID}], \tag{14}$$

where α_I is the angle between the invader's heading and the goal, d_{IG} is the distance between the invader and the goal, β_I is the angle between the invader's heading and the defender, and d_{ID} is the distance between the invader and the defender. In addition, the defender's inputs are as follows,

$$\text{Defender's Inputs} = [\alpha_D \quad d_{DG} \quad \beta_D \quad d_{DI}], \tag{15}$$

where α_D is the angle between the defender's heading and the goal, d_{DG} is the defender's distance to the goal, β_D is the angle between the defender's heading and the invader, and d_{DI} is the defender's distance to the invader.

IV. Proposed Method

In this proposed method, both layers use the FACL algorithm. Fig. 3 shows a diagram of the reinforcement learning structure. Algorithm 1 outlines the training method of the hierchaical learning switch.

A. The Lower-Level Policy

The LLP is where the agents play the game through their primitive actions. In the cooperative game, the agents learn what force to output into the system in order for them to coordinate themselves to get to the end of the hallway. In the competitive game, the invader learns to output a heading required to get to a territory while also not getting captured by the defender. The defender learns to avoid the invader to get the target by intercepting the invader. In order to do this the agents train using the FACL algorithm. A single scenario of a game played is called an episode. Within an episode, the agents play the game to completion, this can either have learning on or off.

B. Higher-Level Policy

The goal of the HLP is to determine when the LLP should begin learning and adapting the actors' and critics' parameters ω^ls and ζ^ls. To train the HLP, the FACL algorithm is used. The output of the actor is the state of the learning switch for the lower level. A positive output means that the learning will switch on in the lower level, and a negative value corresponds

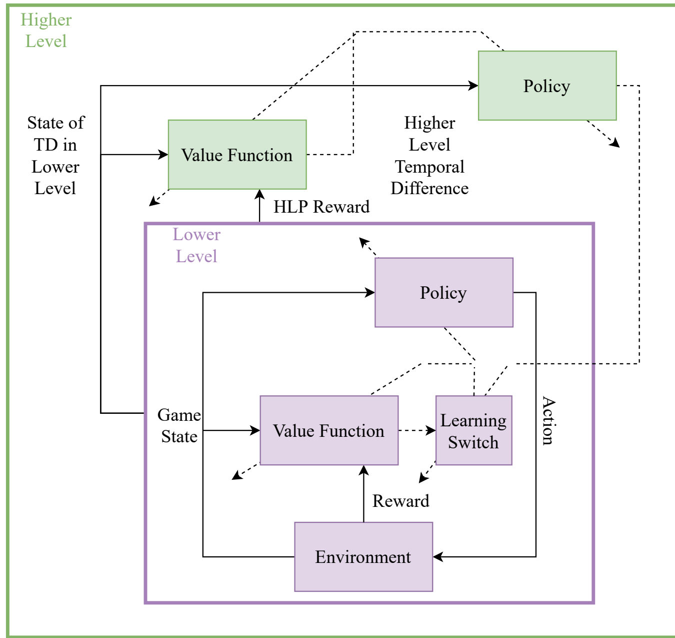

Fig. 3. Model of the Hierarchical Reinforcement Model using FACL

to turning the learning switch off for the next episode to be played.

In order to train this level, a pre-existing LLP will be used. After each episode, the mean and the standard deviation are calculated from all the TDs from each iteration of the episode. These statistics are used as inputs to the higher level to determine if learning is required. The temporal difference is also called the prediction error, if the prediction error suddenly increases in the lower level, this means that some of the output parameters, ω^l, of the lower level policy must be adapted. Based on the statistics of the last episode, the output of the HLP actor will turn the learning on or off for the next episode.

The mean of all the temporal differences is calculated as follows,

$$\mu = \frac{\sum_{t=0}^{T} \Delta_t}{T}. \tag{16}$$

In (16), T is the total number of iterations in the episode and Δ is the temporal difference calculated by (7). The standard deviation of the temporal differences is calculated by (17).

$$\sigma = \sqrt{\frac{\sum_{t=0}^{T} (\Delta_t - \mu)^2}{T-1}}. \tag{17}$$

Thus, the HLP's actor and critic update their policies after each learning episode. The reward used for the HLP is (18), where a and d are constants specified by the user.

$$R = e^{-\frac{\mu^2}{a^2}} + e^{-\frac{\sigma^2}{d^2}}. \tag{18}$$

Algorithm 1 The proposed method algorithm

Initialize:
Set the hyper-parameters.
Import a pre-trained LLP. The LLP was trained via the training loop below, while k_is are set to 1 (k_i=1 means the learning is on).
Initialize the HLPs' actors and critics to be zero vectors for each agent.
Create a vector called TD for storing temporal differences for each agent.
Training Loop:
for Iteration number=1 .. Maximum Iteration **do**
Set an initial location for each agent.
The HLP returns k_is, which are either 0 or 1. Each i corresponds to a single agent.
while t $\leq$ Maximum simulation time OR the agent is in the terminal state **do**
The LLP returns an action for each kind of agent.
The actions are taken and the agents move to the new state.
The reward for the LLPs are received.
Temporal differences are calculated for the LLPs and stored in TD.
for i=1..Number of agents **do**
if $k == 1$ **then**
Update the actor and the critic of the ith LLP via the received reward.
else if $k == 0$ **then**
Do not update the LLP of the ith agent.
end if
end for
end while
The reward for the HLPs are calculated via the temporal differences stored in TD of the game that was just played. In some cases, implementing a filter on TD is beneficial.
Update the actors and the critics of the HLPs.
if Iteration number modulus 500 == 0 **then**
Change environment dynamics.
end if
end for
Finalization: Store the policies.

The reward function shows that a higher reward is given when the mean and standard deviation of the temporal differences within an episode are closer to 0. The benefit of using the TD is the independent nature of the parameter from the chosen game. The same approach was used for both games, although the games are completely different.

V. Results

A. Cooperative Game

The initial LLP was first trained for 50,000 episodes. The LLP is stored as the pre-trained policy. The HLP was then trained for 80,000 episodes. During training, the mass of the

agent and the coefficient of friction would change every 1000 episodes. Each episode runs for 30 seconds. The time step for the simulation is set to 0.01 seconds. This game uses seven triangular membership functions per input for a total of $7^4 = 2401$ rules. The membership functions are evenly distributed in the input interval. The critic learning rate is $\alpha = 0.5$, and the actor learning rate $\beta = 0.3$. The exploration-exploitation factor σ in (2) is set to 0.9. The discount factors for both agents called Diana and Sharon are set to 0.995. For the HLP, the discount factor is set to 1. Seven evenly distributed triangle membership functions are implemented for each input, and σ is set to 0.5.

Every 1000 iterations, the environment of the game will change arbitrarily through the use of a random number generator. In the cooperative game, the mass of the agents can change between 0 and 3 kg, along with the friction coefficient of the hallway. These environment changes impact the coordination of the agents and thus they must adapt their policies to fit the new environment.

Here we show two examples, the first result of this system is shown in Fig. 4. This figure shows three distinct plots. The top plot shows the standard deviation of TDs that were calculated by a single agent within 1 episode. A higher standard deviation implies a wider spread of temporal differences. The middle plot shows the mean of all the temporal definitions calculated within a single episode. The bottom plot shows two important pieces of information, the state of the switch and the successfulness of the episodes. The blue line shows the rate of success within the last 100 episodes. Each red point plotted at 1 represents when the learning switch was set to on, and each red point plotted at 0 represents the learning switch set to off.

Fig. 4 shows that the initial LLP was not trained well enough and thus the HLP turns the learning on. This is shown in the third plot of Figure 4 using the red points plotted at y=1. The blue success rate line in this bottom plot varies. At episode 2000 the mass and friction in (9) are changed which leads to a large decrease in successful episodes with the higher-level switching learning on. The learning switch stays on for 5 episodes, at this point the success rate jumps back to 100%.

Fig. 5 shows a zoomed in plot of Fig. 4 from episodes 4500-10000; we can see how fast the LLP is able to adapt. It is evident that the temporal difference statistics change when the environment changes after every 1000 episodes. The standard deviation of the TDs within the episode shoots up when the environment changes; when this occurs the HLP signals to turn learning on within the LLP. The Switch State and Success Ratio Plot shows how quickly the policy adapts once the HLP switches learning on. For example, after episode 5000 the environment changes, it took 4 episodes with the learning on for the success rate to climb back up to 100%. The learning first switched to on at episode 5004. When the environment dynamics changed once again after episode 6000, the higher-level turned learning on immediately after the first unsuccessful episode. This unsuccessful episode with the new dynamics had the standard deviation of TDs within the episode

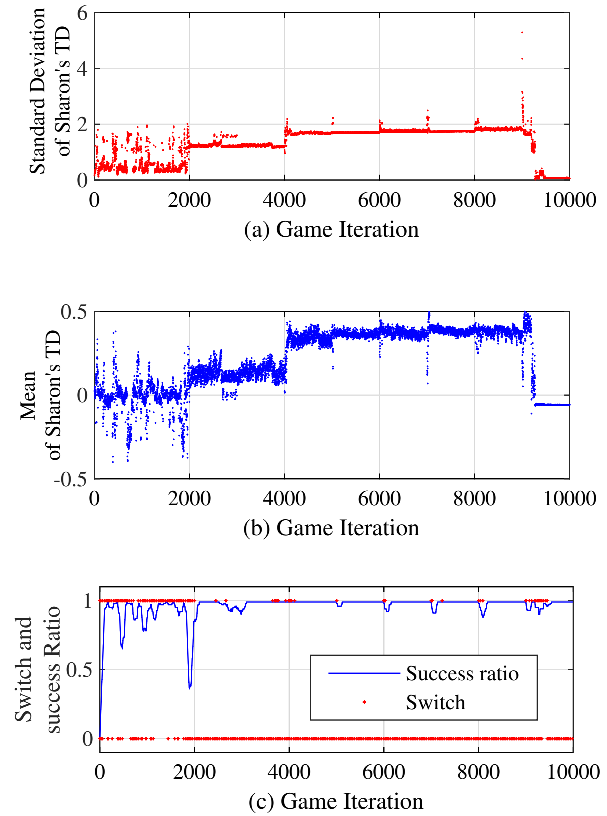

Fig. 4. (a) The standard deviation of TD for agent Sharon. (b) The mean of TD for agent Sharon. (c) Success ratio and the state of switch (0: off, 1: on) for agent Sharon.

jump to 2.07. The higher-level policy was able to use this data to switch the learning on. Once the learning was switched on, it stayed on for 17 episodes. During these 17 episodes, the agents adapted only the applicable output parameters ω^l of their lower-level policies. Similar trends appeared during environment changes taking place at episodes 7000, 8000, and 9000; with 9000 being a more extreme change in values.

Fig. 6 shows another example of using hierarchical learning for the same cooperative game. In this example, the environment changes 3 times, at the 500, 1000 and 1500 episode mark. In this example, the initial LLP used is already perfect and does not require further learning. This policy was trained with m=1 Kg, and b=0.1 in (19). The first 500 episodes played shows a perfect score seen at 100% in Fig. 6, the higher-level learning was never switch on. At episode 501, the coefficient of friction, b, changes from 0.1 to 0.01. At episode 514 the learning switch turns on. There are 3 episodes total that are played with learning turned on. This shows the robustness of the pre-trained policy. At episode 1001, the mass suddenly decreases from 1kg to 0.5kg along with a slight increase in the friction coefficient, b, to 0.05. The very next episode, the HLP is turns learning on for the LLP. We can see how the standard deviation of TD jumps up and the mean of TDs becomes

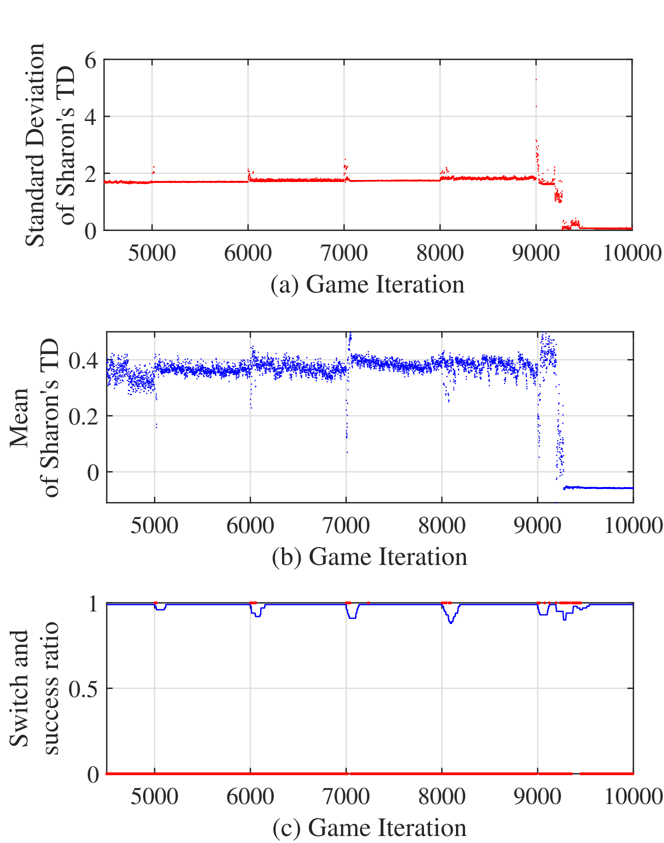

Fig. 5. Episode 4500 to 10000 zoomed in of Fig. 4

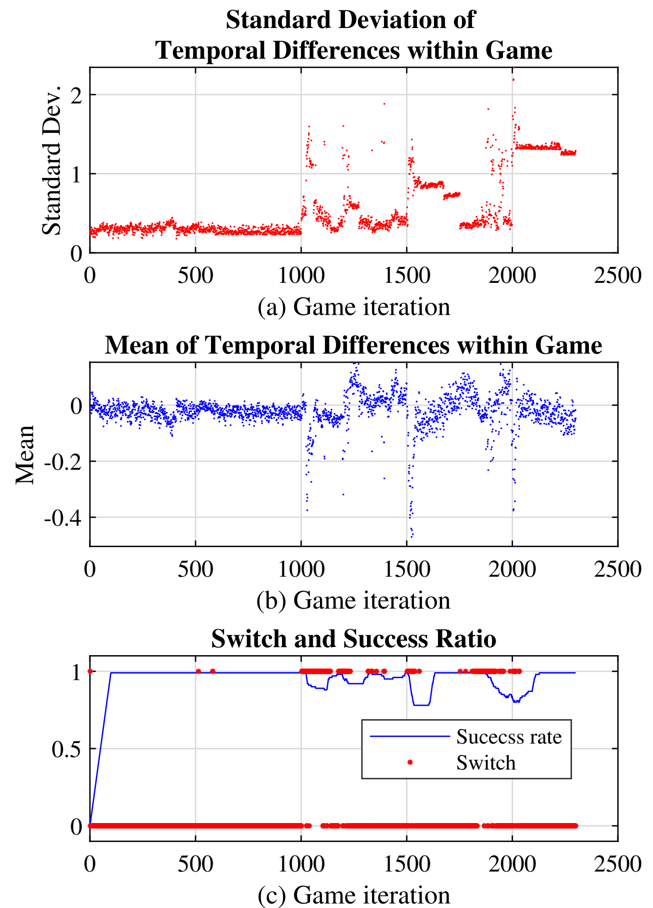

Fig. 6. Second example of the cooperative game, continuous hallway

negative. It takes 184 episodes for the LLP output parameters to adapt to this environment change. The standard deviation of TD settles to approximately 0.3. Finally, at episode 1501, the environment changes in a greater manner, with the mass increasing to m=1.4kg and keeps the friction coefficient at b=0.05. Once again there is a large drop into the negative mean of TD along with a spike in standard deviation of TD. We can see there are two periods of intense learning that take place by looking at the blue line success rate plot. Learning in the LLP occurs on and off until it finally settles on a new policy at episode 2034. The total number of episodes that used learning were 134.

We can clearly see in both Figs. 4 and 6 that the standard deviation and mean of the TD being recorded in each episode spikes during an environment change. More extreme changes require more learning episodes to adapt the output parameters.

B. Competitive game

The game of guarding a territory, which was presented in section III-C is implemented to tackle the problem in the competitive game. The nature of the competitive games are different from cooperative games. Thus, reaching to a point where success rate will be 100% is not desirable. Instead, the game has to converge to some point, which is called the *Nash equilibrium*. In the game of guarding a territory both agents strive to win the game. Let us elaborate on the topic with an example. In a field, where the invader and the defender have equal speed and maneuverability, depending on the location of the goal, the chance of each agent winning the game varies. The invader wins the game if it has a less distance to the goal. The defender wins if it is closer to the goal than the invader. However, by assuming arbitrary locations for the invader and the defender and the goal in numerous games, the chance of each agent winning the game will be equal. On one hand, giving a higher speed to the invader increases its chance of winning. On the other hand increasing the maneuverability of the defender increases the defender's chance of winning.

To train the invader and the defender, we set the discount factors γ to 0.7, the critics' learning rates $\alpha = 0.1$, the actors' learning rates $\beta = 0.05$, and $\sigma = 0.5$. The hyper-parameters are not decaying over the time, so the environment changes could be reflected to the LLP output parameters. The maximum simulation time for an episode is 100 seconds, and time step is set to 0.1 seconds. In addition, 11 triangular membership function are selected for each input. The membership

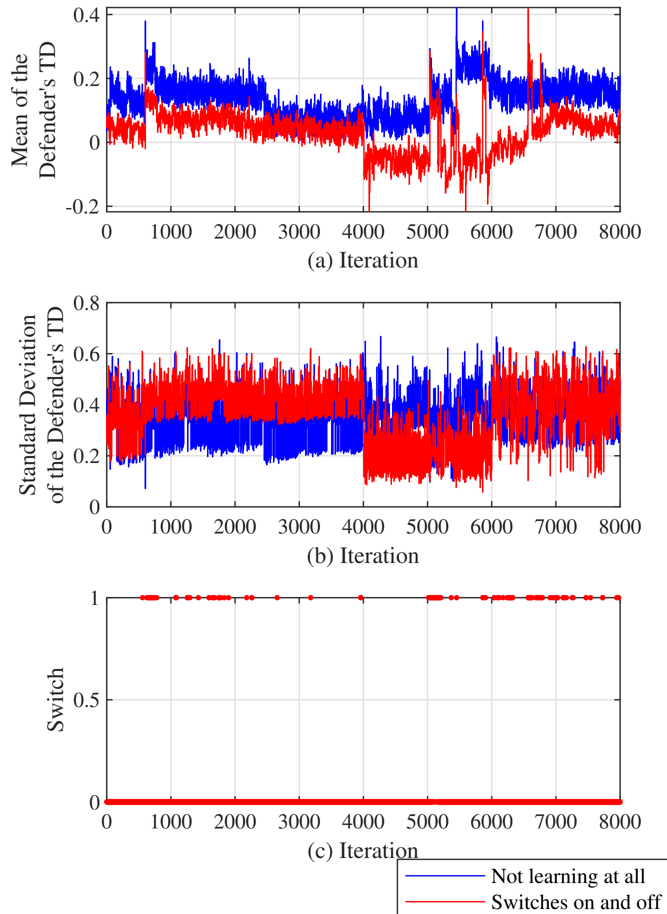

Fig. 7. (a) The standard deviation of TD for the defender. (b) The mean of TD for the defender. (c) The state of switch (0: off, 1: on) for the defender.

functions are evenly distributed in the input intervals. The LLP is trained for 50,000 iterations initially. The discount factor for the HLP is set to 0.995, the actors' learning rate is set to 0.5, the critics' learning rate is set to 1.0, and σ is set to 0.5. The actors' and the critics' membership functions are triangular shaped and have 11 membership functions each. Then, the HLP was trained for 50,000 iterations, where in every 1,000 iterations, the environment changes.

In the competitive game, the environment change occurs by changing the speeds and maneuverability. The speeds change between 0.6 to 1.4 units/sec, and parameter L in (9) changes between 0.25 to 1.5 units.

Unlike the cooperative game in section V-A, in this section success rate of the agents is not telling. The reason is each kind of agent tries to increase its success rate. So, a success rate of 100% is not desirable. Instead, we show the mean and the standard deviation of the defender's temporal difference, in cases where the learning is off during learning and switching learning.

Figure 7 shows that by turning the learning switch on several times during the learning process, the mean and the standard deviation of the defender are significantly lower.

VI. Conclusion and Future Works

In this paper we proposed a hierarchical reinforcement model that learns when a non-stationary environment changes and allows the agent to adapt their policies to reflect the new environment. In this model, the HLP switch sends a signal to turn learning on in the LLP. Once the network parameters have successfully adapted to the new environment, the HLP sends a signal to turn the learning off. The HLP is trained using temporal difference statistics from the LLP as an input, along with environment changes that happen every 1000 episodes. The HLP's reward function is to minimize the standard deviation and mean of the temporal differences calculated within an episode.

The temporal differences within an episode of a game can indicate a change in the game environment. By looking at the statistics of the temporal differences within an episode, it becomes clear how the temporal difference can be used as a key indicator. We show that more extreme environment changes require more episodes with training on in order to adapt the output parameters of the policy.

The contributions that are presented are 1) a method using the temporal difference to turn the learning on and off in a non-stationary environment. 2) We examined the proposed method using a multi-agent cooperative game and a competitive game.

References

[1] S. Padakandla, P. KJ, and S. Bhatnagar, "Reinforcement learning algorithm for non-stationary environments," *Applied Intelligence*, vol. 50, pp. 3590–3606, 2020.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] H. Schwartz, "An object oriented approach to fuzzy actor-critic learning for multi-agent differential games," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 183–190, IEEE, 2019.

[4] A. Asgharnia, H. Schwartz, and M. Atia, "Learning multi-objective deception in a two-player differential game using reinforcement learning and multi-objective genetic algorithm," *International Journal of Innovative Computing, Information and Control*, vol. 18, no. 6, pp. 1667–1688, 2022.

[5] A. Asgharnia, H. Schwartz, and M. Atia, "Learning deception using fuzzy multi-level reinforcement learning in a multi-defender one-invader differential game," *International Journal of Fuzzy Systems*, vol. 24, no. 7, pp. 3015–3038, 2022.

[6] M. O. Karabag, M. Ornik, and U. Topcu, "Deception in supervisory control," *IEEE Transactions on Automatic Control*, vol. 67, no. 2, pp. 738–753, 2021.

[7] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[8] R. Everett and S. J. Roberts, "Learning against non-stationary agents with opponent modelling and deep reinforcement learning.," in *AAAI Spring Symposia*, 2018.

[9] Y. Qin, T. Menara, S. Oymak, S. Ching, and F. Pasqualetti, "Non-stationary representation learning in sequential linear bandits," *IEEE Open Journal of Control Systems*, vol. 1, pp. 41–56, 2022.

[10] X. Deng, Y. Zhang, and H. Qi, "Towards optimal hvac control in non-stationary building environments combining active change detection and deep reinforcement learning," *Building and environment*, vol. 211, p. 108680, 2022.

[11] L. Jouffe, "Actor-critic learning based on fuzzy inference system," in *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, vol. 1, pp. 339–344, IEEE, 1996.