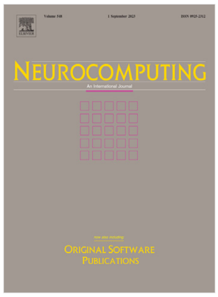
Journal Pre-proof



An adaptable fuzzy reinforcement learning method for non-stationary environments

Rachel Haighton, Amirhossein Asgharnia, Howard Schwartz, Sidney Givigi

PII: S0925-2312(24)01080-4
DOI: https://doi.org/10.1016/j.neucom.2024.128309
Reference: NEUCOM 128309

To appear in: *Neurocomputing*

Received date : 11 November 2023
Revised date : 3 July 2024
Accepted date : 1 August 2024

Please cite this article as: R. Haighton, A. Asgharnia, H. Schwartz et al., An adaptable fuzzy reinforcement learning method for non-stationary environments, *Neurocomputing* (2024), doi: https://doi.org/10.1016/j.neucom.2024.128309.

An Adaptable Fuzzy Reinforcement Learning Method for Non-Stationary Environments

Rachel Haighton†, Amirhossein Asgharnia†, Howard Schwartz, Sidney Givigi

Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, K1S 5B6, Ontario, Canada.

*Corresponding author(s). E-mail(s): rachelhaighton@cmail.carleton.ca;
Contributing authors: amirhosseinasgharnia@cmail.carleton.ca; schwartz@sce.carleton.ca; sidney.givigi@queensu.ca;
†These authors contributed equally to this work.

Abstract

How do we know when a reinforcement learning policy needs to adapt? In non-stationary environments, agents must adapt and learn in environments that change dynamically. We propose a finite-horizon model-free solution using a hierarchical learning structure with fuzzy systems. The higher-level learning policy advises the lower-level policy when to start and stop learning based on the temporal differences calculated within the lower-level. Major differences in the temporal difference of each action produced by an agent may indicate environment change. This structure is tested with multi-agent differential games in both the cooperative and competitive aspect. Results show that this method is quick to notice and adapt the policy within relatively few learning episodes.

Keywords: reinforcement learning, non-stationary environment, multi-agent system, fuzzy systems

1 Introduction

Learning and adapting the parameters of a network controller of an autonomous system can be computationally expensive. Once the parameters are properly converged, it is important to stop the learning; 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 can occur repeatedly is often noted as a Non-Stationary Environment. This type of an environment requires near constant learning or more sophisticated methods to ensure proper adaptation.

Fuzzy inference systems are excellent non-linear function approximators which makes them ideal for the actor-critic learning scheme in reinforcement learning. A major strength of fuzzy systems is their ability to evolve their parameters. A lot of autonomous fuzzy systems research look at using recursive least squares techniques for consequent parameter learning. Instead, we use a reinforcement learning actor-critic structure to do this where the value function and actor can be approximated and adapted with fuzzy inference systems.

Within the structure of actor critic learning, a term called the temporal difference (TD) is calculated and used to update the consequent parameters of each rule. The temporal difference error can be defined as a way to quantify the degree of temporal inconsistency between estimates made by the value function. The temporal difference can be thought of as the prediction error; by using this prediction error as an indicator of the state of the environment, it provides a knowledge of when the FIS must adapt to the new changes in the non-stationary environment.

In order to determine and keep track of when the FIS controller must be updated to be compliant with the environment, a second FIS controller is trained and used with the temporal differences of the agents as an input. This secondary controller simply determines when the main controller needs to re-adapt to the new environment and when it can stop adapting the consequent parameters for the rules. This hierarchy allows for a more seamless adaptation based on temporal difference statistics that occur within the FIS controller of the agents. Other non-stationary environment learning models use a secondary model to determine changes such as [1] and [2] however they are much more computationally expensive and complex. By combining the concepts of adaptable fuzzy systems with temporal difference reinforcement learning, a hurdle is removed for autonomous systems in non-stationary environments. Another possible approach to detect changes in the environment was to watch the game's result. By observing a significant change in the result, we could trigger re-training. However, in such a method, the higher-level policy, which decides about the re-training would be dependant on the game's nature. To circumvent this issue, we used TD, which is a universal metric in all applications that are trained via reinforcement learning. The fact that the TD is a signal opens a place to use its properties, such as derivative to predict future changes as well. It should be mentioned that the proposed method addresses finding significant changes in the environment. The games and the algorithms are designed in a way that are robust to small uncertainties such as asynchronously in the state updates and real-time processing constraints.

In this paper, we used differential games as a platform for our simulations. Differential games are games that follow differential equations. Although most of applications in the literature are about pursuit-evasion games, they have many other applications in prediction, economics and sports [1].

1.1 Contributions

The contributions of this paper are threefold and include:

- A study of multi-agent systems in non-stationary environments. By studying how the statistics of the temporal difference from an agent react during environment changes, we see that it is a good indicator of when a policy should or should not adapt.
- A model-free finite horizon method for learning in non-stationary environments. This method uses an evolving fuzzy system that uses reinforcement learning for consequent parameter adaptation.
- This paper strengthens the importance of using fuzzy systems in the field of reinforcement learning. Since fuzzy approximators allow for easy interpretability in machine learning, they act as a more transparent method in the field of artificial intelligence. This paper does an analysis of consequent rule parameters during adaptation which is simple since each rule corresponds to an observed state in the system.

1.2 Organization of Paper

The rest of the paper is organised as follows. In Section 2, we discuss related work including other methods to do with reinforcement learning in non-stationary environments, and fuzzy reinforcement learning. Section 3 looks at the differential games used to demonstrate the proposed method. Section 4 is dedicated to the tools and concepts that we used for our proposed method. Section 5 describes the hierarchical learning switch that this paper proposes as a solution to reinforcement learning in non-stationary environments. We describe how to train and execute the learning switch. Section 6 displays and analyzes the results of the proposed method for the cooperative and competitive games.The environmental changes and the proposed methods are studied at the rule level with different examples given from each game. Finally, section 7 concludes the paper with future works and references.

2 Literature Review

In this paper we address the problem of learning in non-stationary environments. We use differential games as the learning environment. Differential games are the generalized form of game theory problems, where the state and action space are continuous [2]. In other words, the agents are governed by differential equations. In differential games, the players may cooperate to accomplish a common goal, or the players my compete against each other. Thus, we can divide differential games into two groups of cooperative games, and competitive games. In cooperative games, the agents have a common goal and try to maximize the pay-off of the whole group in the game. In competitive games, the pay-offs of the agents are conflicting: if an agent gets more reward, the other agent loses reward.

In [3], the authors tackled a pursuit-evasion game using a genetic algorithm. This paper is among the first papers that provide an artificial intelligence solution for the game in a virtual reality environment. However, the method has some limitations. To use population based optimization algorithms, one needs to simulate the game several times with the same environmental variables, such as initial conditions. Thus, the policy found by the proposed method suffers if a the game starts from a new initial point. This problem is also addressed in [4], by using several initial conditions for each cost function evaluation. The cost function was defined to be the average of several cost functions with different initial position for the robots. The initial positions were distributed on the game field's boundary.

In [5], the authors modeled an N-pursuers M-evaders game. In the proposed game, the evaders are omniscient and they do not have limitations in the speed. The evaders act like a contaminating gas in an unknown environment. However, the pursuers have limited speed, information about the environment map, and sensor range. The authors initially used the genetic algorithm to evolve suitable control policy for the pursuers. However, the policy could not handle a general case, where the initial condition or map was different. To address this limitation, the authors propose a complementary approach in which a random walk is used alternatively with the evolved automaton, indicating random actions in cases of states not sufficiently visited during evolution.

Another way to generalize the policy is to use a reinforcement learning strategy. Reinforcement learning algorithms, such as Q-learning are independent of the initial condition. In addition, unlike the population based optimization algorithms, by using a reinforcement learning algorithm, only those variables that have a effect on the outcome get credit from the algorithm [6]. However, the major drawback of using a reinforcement learning algorithm is defining a suitable reward function as well as hyper-parameters [7]. The type of reward functions that are used in pursuit-evasion games are instantaneous, which means the reward signal is given to the agent at each time step [4, 8]. These reward functions are weighted, and the weight is dependent to the game's environment. Changing the game environment not only makes the policies unfit, but also hinders the relearning process because of the wrong reward functions.

The idea of changing environments in autonomous systems has been well established. Concepts of dynamically changing data patterns (data drifts) have been studied in fuzzy systems for over two decades [9]. Often this research is broken up into two schemes: evolving the structure of the FIS and updating the parameters. In this paper, we focus on updating the parameters to reflect the dynamically changing environment. Work that looks at parameter learning schemes for first-order fuzzy rules often use recursive weighted least square techniques such as [10]. Other parameter tuning schemes also included Extended Kalman Filter based techniques [11] and gradient descent-based techniques [12]. The idea of using reinforcement learning to adapt fuzzy parameters in a dynamically changing environment is novel.

A survey that focuses on reinforcement learning in non-stationary environments was produced by [13]. Within the survey the author categorizes the proposed solution by model-based or model-free and finite horizon/infinite horizon approaches. Since we are focusing on differential games, the approach is finite horizon. [14] is also an online model-free solution; it focuses on minimizing a regret function. The regret is defined as the difference between the average reward per step and the average reward obtained by the best stationary deterministic policy. The algorithm separates the learning iterations into intervals and within each interval, the Q values are learnt from the reward samples of that interval. Another regret-based approach is [15]. This model-based method does not scale well to large state-action space MDPs.

The authors in [16] introduce a new class of graphical model that allows for the conditional dependence structure of data-generation processes to change over time. This framework has numerous applications, from studying transcriptional regulatory networks during an organism's development to analyzing traffic patterns throughout the day. The authors propose a Markov Chain Monte Carlo (MCMC) sampling algorithm to learn the structure of non-stationary dynamic Bayesian networks.

Ref. [17] presents data-dependent learning bounds for the general scenario of non-stationary non-mixing stochastic processes. The key ingredients of the generalization bounds are a data-dependent measure of sequential complexity and a measure of discrepancy between the sample and target distributions. The learning guarantees presented in the paper hold for both bounded and unbounded memory models, covering the majority of approaches used in practice, including various autoregressive and state space models. Therefore, the paper was able to learn in a non-stationary environment by using a data-dependent approach that takes into account the complexity of the process and the discrepancy between the sample and target distributions.

Another method exits where the learning rate adapts during the training phase. Examples of this method include [18]. While the method presented in [18] has been successful in non-stationary environments, it is computationally expensive since the learning algorithm must always be on, regardless of how small the updates to the network are.

Fuzzy learning systems has been used in a variety of applications over the years including [19], [20], [21]. Using fuzzy inference systems as function approximators often make the system computationally simpler while also increasing the interpretability of the results. Having interpretable and transparent results is becoming a much larger issue in the field of machine learning which fuzzy systems handle quite well.

The idea of using temporal differences as an indicator of policy was described in [8]. By studying the statistics of the temporal differences that occur while learning a policy, there is confidence that adaptation is converging to the correct values. This paper is an extension of the idea presented [8]. One possible way to detect the changes in the environment is to observe the game's outcome. However, since the definition of a successful game varies among different games, a better option is to choose to watch the TD as the change indicator as suggested in [22].

3 Differential Games

Three different games are presented in this section. These games are of the multi-agent learning realm and are used to test the learning switch. There are two categories of games used, cooperative where agents must cooperate to complete a task; and competitive where agents are in a adversarial scenario.

3.1 Cooperative

There are two cooperative games used, one called continuous hallway and another called balancing a ball.

3.1.1 Continuous Hallway

Based on a game presented in [23], two agents are randomly placed in a hallway and must reach the end of the hallway at the same time. The agents must coordinate their actions based on distance and velocity inputs. In the continuous version, agents are rewarded a terminal reward of +3 for getting to the end of the hallway, but if both agents get to the end of the hallway at the same time then they are both rewarded +15. Fig. 1 shows an illustration of this game.

In this game the environment changes include the mass of the agents, m, and the coefficient of friction, b. The dynamics of the agents are given by 1.

$$m\ddot{x} + b\dot{x} = F, \tag{1}$$

where b is a coefficient of friction and m is the mass. The shaping reward is as follows,

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

where w represents a weight. The weight used is 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 other agent. The shaping reward indicated to the agent that getting closer to the end of the hallway is generally more important than staying close to the other 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. Additionally, D represents a function of distance. For example, $D_{ig}(t)$ represents the distance between a location of agent i, and the location g, the goal or end of the hallway at time t. D_{ij} is the distance between agent i and agent j.

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. There is no communication method between the agents; the agents must only make decisions based on velocity and distance information. The output of the agent is the force, F in (1) which has a maximum of +3N and minimum of -3N. Since this is generated by a Fuzzy Logic Controller (FLC) it is a continuous action space.

This game uses 7 membership functions per input for a total of $7^4 = 2401$ rules. The critic learning rate is $\alpha = 0.5$ and the actor learning rate $\beta = 0.3$.

To differentiate between the two agents, one was named Diana and the other was named Sharon. This was done for clarity in the results and discussion section.

3.1.2 Balancing A Ball

This game was inspired by [24] where two robots on either side of a 2D table and they must adjust the heights of either end of the table to balance the ball in the middle of the table. An illustration of this game can be found in Fig. 2.

The goal of the agents is to adjust the table to balance the ball in the middle. An agent can only adjust the height of its end of the table as illustrated in Fig. 2. The game ends when the ball rolls off the table or 10 seconds has passed. The dynamics of

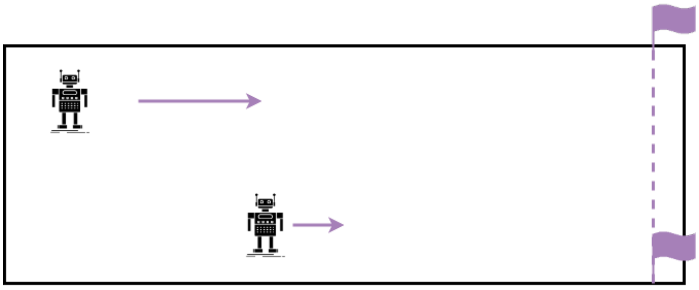


Fig. 1 Illustration of Continuous Hallway

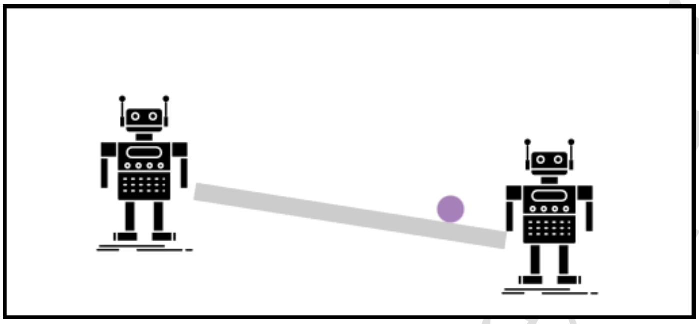


Fig. 2 Illustration of Balancing A Ball game

the ball is 3 where m represents the mass of the ball, c is the friction coefficient, g is the gravitational coefficient, l is the length of the table, h_1 and h_2 are the heights of each end of the table.

$$m\ddot{x} = -c\dot{x} + mg * ((h_1 - h_2)/L), \tag{3}$$

There are two inputs into the controller of each agent: the position of the ball on the table, and the velocity of the ball. In each episode the ball starts randomly on the table with a random position and velocity. The agents output a height which corresponds to the height of their side of the table. This action can tilt the table if one side is higher or lower than the other side. This action space is continuous but capped at $+/-1m$.

There are 15 membership functions for each of the two inputs with boundaries between -3m and +3m for the position, and -4 m/s and +4 m/s for the velocity of the ball. The critic learning rate is $\alpha = 0.1$ and the actor learning rate $\beta = 0.05$.

The reward function used to train the policy of each agent is (4) where x is the position of the ball on the table, with the center of the table being $x = 0$, and $\dot{x}$ is the velocity of the ball. The reward function gives larger rewards when the position of the ball is closer to the center of the table ($x = 0$), and when the velocity is also close to zero $\dot{x} = 0$.

$$r(t+1) = 0.8e^{-x^2/0.25} + 0.2e^{-\dot{x}^2/0.25} \tag{4}$$

3.2 Competitive Games

3.2.1 Pursuit-Evasion Game

Pursuit-evasion (PE) games are a class of differential games, where the participating agents have conflicting interests. There is a group of agents called invaders that want to reach a target. There is another group of agents called defenders, and they want to capture the invader and defend the target [25]. In a PE game the target may be stationary, or it also moves in the game field. The kinematics of each agent is given by the differential equation that describes a car motion as,

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v\tan(\varphi)}{L} \end{cases}, \tag{5}$$

where (x, y) is the agent's location. The term θ is the agent's heading with respect to the x-axis. The term φ is the steering angle of the agent. The steering angle is the output of the agent, this action is from a continuous action space with values between $\frac{\pi}{4}$ and $\frac{-\pi}{4}$. The terms v and L are the agent's speed, and distance between the forward and rear axles, respectively.

The game finishes when at least one invader reaches the target, or the defenders capture all the invaders. In this paper, we assume there is one invader and one defender and one target.

The invader's reward function, R_{inv}, and the defender's reward function, R_{def}, are shown as follows,

$$\begin{aligned} R_{inv} &= W_I(d_{IG}(t) - d_{IG}(t+1)) + (1 - W_I)(d_{ID}(t+1) - d_{ID}(t)) \\ R_{def} &= W_D(d_{ID}(t) - d_{ID}(t+1)) + k(1 - W_D)(d_{DG}(t) - d_{DG}(t+1)). \end{aligned} \tag{6}$$

In (6), $d_{IG}(t)$ is the Euclidean distance between the invader and the goal, $d_{ID}(t)$ is the Euclidean distance between the invader and the defender, $d_{DG}(t)$ is the Euclidean distance between the defender and the goal. The terms W_I and W_D are called the invader's and the defender's reward weights, and they weight one term of the reward function over the other one. The parameter k in (6) is set to zero if the defender is getting further from the goal and otherwise it is set to 1 [25].

To return an action, the invader's and the defender's policy need to take in inputs. The invader's and the defender's policy inputs are,

$$\begin{aligned} \text{Invader's Input} &= [X_I(t) \quad Y_I(t) \quad \theta_I(t) \quad X_D(t) \quad Y_D(t)] \\ \text{Defender's Input} &= [X_D(t) \quad Y_D(t) \quad \theta_D(t) \quad X_I(t) \quad I_D(t)], \end{aligned} \tag{7}$$

where $(X_I(t), Y_I(t))$ is the invader's Cartesian location, $\theta_I(t)$ is the invader's heading with respect to the x-axis, $(X_D(t), Y_D(t))$ is the defender's Cartesian location, $\theta_D(t)$ is the defender's heading with respect to the x-axis.

Unlike [4, 25], the agents of the competitive game are not omniscient and they do not have a complete vision of the goal location. The agents have to find the goal and build their policy based on the discovered target. This means after changing the goal location, the learnt policy is not suitable anymore.

4 Fuzzy Actor Critic Learning and Temporal Difference

4.1 Reinforcement Learning

The primary approach to solve the problems in this study is reinforcement learning. Reinforcement learning (RL) is regarded as the third paradigm of machine learning approaches in the literature [6]. RL is different from supervised learning, where a set of examples are provided to train a model. In RL, there is no set of examples and pieces of data, but an agent finds the suitable action via trial and error. RL is also different from unsupervised learning, where a set of unlabeled data are clustered. RL is different from optimization methods, where all the parameters of a model are equally credited, even if they are not used in creating the output in a certain instance [6]. In reinforcement learning, the model returns the best action that maximizes a trade-off between the current outcome and the future outcomes.

Reinforcement learning problems are often modeled as an Markov Decision Process (MDP). An MDP is a tuple (S, A, P, R, γ), where S is the set of states, A is the set of actions, given each state, $P(s, a, s')$ is the probability of reaching state $s' \in S$, by taking action $a \in A$ from state $s \in S$. The term R is the reward function, a signal that assess the quality of a taken action. Finally, the term γ is a discount factor and magnifies the current action over the future actions.

In a problem with limited number of states and actions, one may use a table to store the quality index of an action given a state. However, when the number of states and actions is large (but still bounded), or when the game is played in the continuous domain, using a model estimator is useful. In this paper, we used a fuzzy inference system to estimate the quality of each action, given a state.

4.2 Fuzzy Inference Systems

To map a state into an action we implemented a fuzzy inference system. We used the Takagi-Sugeno (TS) fuzzy inference model [26]. Unlike the Mamdani fuzzy inference model, where there are output membership functions, in Takagi-Sugeno model, the output parameters are all singletons. The output is calculated by the linear combination of the output parameters. There are two advantages for using a TS model in this paper: they have less computational complexity, and they can be adapted in a more straight forward fashion. Because of the latter advantage, it is easier to combine a TS fuzzy inference model with an RL paradigm.

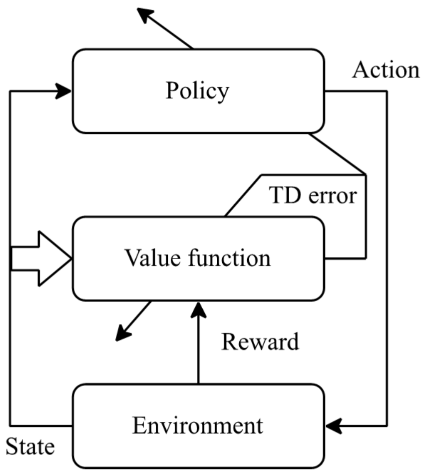


Fig. 3 A diagram of the FACL algorithm. There are two main components: the actor and the critic. They are both approximated with their own fuzzy inference system.

4.3 Fuzzy Actor Critic Learning

The Fuzzy Actor Critic Learning algorithm (FACL) is a reinforcement learning method that uses fuzzy systems as function approximtors for the actor and the critic. The actor acts as the controller or policy and and the critic estimates the value of the current state. The value function is used to calculate the prediction error or the temporal difference. Fig. 3 shows a diagram of the structure of FACL. The algorithm was initially proposed in [27], as a tool to map a continuous input state to a continuous action. More specifically, a Takagi-Sugeno (TS) fuzzy logic controller (FLC) is utilized for the actor; and a TS fuzzy inference system (FIS) is used for the critic which estimates the state value function. The output of the actor is given by (8) 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. Since the fuzzy rules have meanings with respect to the inputs provided, when a policy is no longer sufficient for some states, the algorithm will only adapt the consequent parameters for the rules that fired instead of adapting the entire network. Given we use triangular membership functions only 2^i rules will fire where, i is the number of inputs. The actor output or the control signal is given by 8.

$$u_t = \sum_{l=1}^{L} \phi^l \omega_t^l. \tag{8}$$

When learning the policy, noise is added to the output to mimic exploration. The noise is taken from a normal distribution that has a mean of 0, and a standard deviation σ, noted as $N(0, \sigma)$. We refer to σ as the exploration-exploitation factor, and is largely based on the dynamics of game being played. The control signal during the learning process is,

$$u'_t = u_t + n(0, \sigma). \tag{9}$$

The firing strength of the rule is represented by (10), and is shown as follows,

$$\phi^l = \frac{\partial u}{\partial \omega^l} = \frac{\prod_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i)}{\sum_{l=1}^{L}(\prod_{i=1}^{n} \mu^{F_i^l}(\bar{x}_i))}, \tag{10}$$

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 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:

$$V_t = \sum_{l=1}^{L} \phi_t^l \zeta_t^l \tag{11}$$

$$V_{t+1} = \sum_{l=1}^{L} \phi_{t+1}^l \zeta_t^l \tag{12}$$

where V_t is the value function at time t, ζ_t^l is the critic's output parameter given rule l at time t, and γ is the discount factor. Using the value function, we can estimate the prediction error or temporal difference (TD) as,

$$\Delta_t = r_{t+1} + \gamma V_{t+1} - V_t \tag{13}$$

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.

Using the temporal difference (13), the actor and critic policies are then updated with (14)and (15).

$$\omega_{t+1}^l = \omega_t^l + \beta \Delta_t \phi_t^l (u'_t - u_t), \tag{14}$$

where β is the learning rate of the actor, and Δ_t is the temporal difference, 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.

$$\zeta_{t+1}^l = \zeta_t^l + \alpha \Delta_t \phi_t^l \tag{15}$$

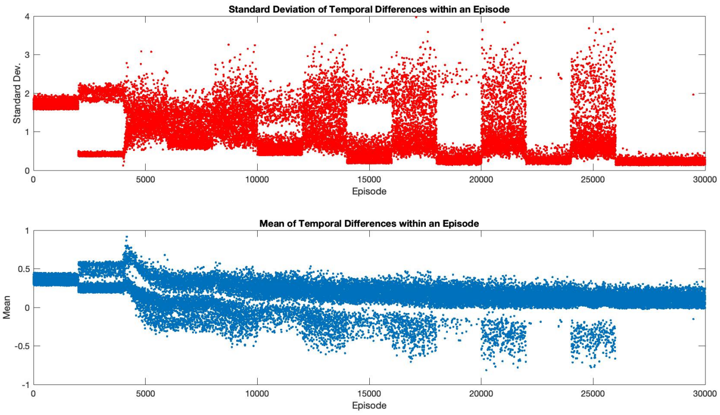


Fig. 4 Temporal difference statistics during while adapting a policy

4.4 Temporal Difference

The temporal difference is an important parameter in the actor-critic structure of reinforcement learning. The temporal difference error aims to quantify the degree of temporal inconsistency between estimates made by the value function in successive time steps. The goal of the value function is to estimate the expected return for an agent at a given state. The temporal difference is then used to update both the critic and the actor. Since the temporal difference acts as a prediction method, the temporal difference can indirectly give important insights to the state of the environment.

To determine the impact of environment changes on the temporal difference, a pre-trained policy had its environment suddenly change. The learning is switched on and off over the required policy adaptation period to see how the temporal differences within a game respond. The pre-trained policy was trained for the continuous hallway game where two agents must meet each other at the end of a hallway without communication. The agents play 2000 episodes with the original policy, in other words, there is learning is turned off. The pre-trained policy or the pre-trained consequent parameters for the actor and the critic were trained based on environment which consisted of a hallway of 15m long, a mass of an agent, m, being 1kg and the coefficient of friction, b, being 0.1. The environment dynamics are given by (16). After these initial games are played, the environment suddenly changes, the agents now have a mass of 0.1kg and the coefficient of friction is changed to 0.01. These changes are large enough for policies to be insufficient and the agents are no longer be able to successfully coordinate.

$$m\ddot{x} + b\dot{x} = F(t) \tag{16}$$

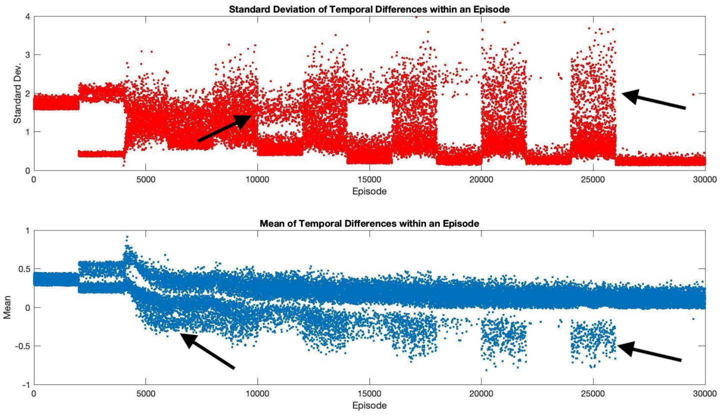


Fig. 5 Temporal difference statistics where fuzzy rule parameters have yet to be learned

Fig. 4 shows two plots, one that shows the mean of the temporal differences of a single agent within a single episode, and one that shows the standard deviation of temporal differences of a single agent within a single episode. Each point represents the statistics within an episode. At episode 2000 the environment changes, and these plots show an obvious impact. The standard deviation and mean plots show a separation of the points and form two distinct bars. At episode 4000, the learning is switched on for both agents and the temporal differences within a game varies a lot while the policy adapts. Recall that during learning the output signal is corrupted by noise used for exploration and results in a spread of the points. High standard deviations and larger magnitudes of means implies that more learning is required. Every 2000 episodes the learning is switched on or off until the policy is converged which occurs around episode 26,000. As the policy adapts to its new environment, the spread of the points decreases. It is visually clear when the learning is on or off based purely on the spread of the points.

Fig. 5 points to a data trend that slowly vanishes with increased policy adaption. What does this data represent? The sporadic points that we see in the standard deviation and the mean are due to both the noise applied during learning but also the learning of new fuzzy rule parameter in the game. These points have rules fired that were never previously fired and are now being learned from 0 due to the new environment dynamics. As these fuzzy rule parameters are learned, these points disappear. These points in the plots shown in Fig. 5 disappear with learning and is seen most clearly from episode 14000 onwards whenever the learning is switched off and these points appear less often. When a policy is initialized, all fuzzy rule parameters are set to zero. If the rules do not fire during training, then they are never updated. However, when the environment dynamics are changed, there is now possibility for the consequent parameters for these rules to now fire due to new states appearing. Since the

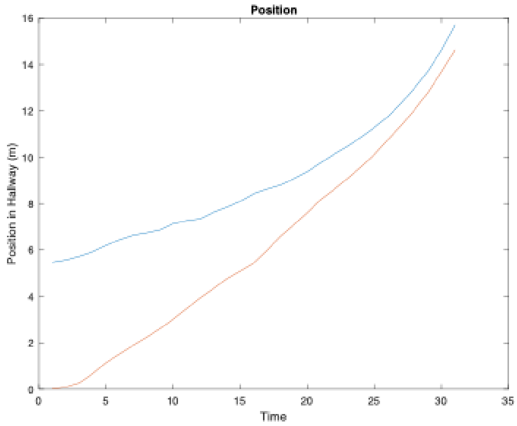


Fig. 6 Position plot of Sharon (blue) and Diane (orange) during a game where new rules are firing.

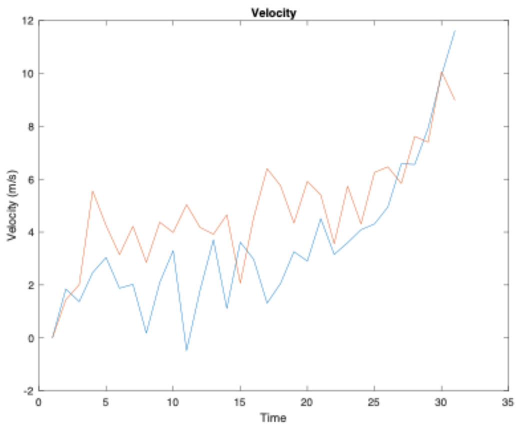


Fig. 7 Velocity plot of Sharon (blue) and Diane (orange) during a game where new rules are firing.

temporal difference error is used as a prediction error, the temporal differences calculated within a game may become large signifying larger errors in a given state. This in turn will increase the standard deviation of temporal differences within a game and impact the mean.

Figs. 6 and 7 shows a single episode from one of the points the arrows are pointing at in Fig. 5. This game has the agents begin in the hallway at the 0.012m mark and the 5.46m mark. The temporal differences that were calculated in this game from a single agent show values from -11.86 to 6.1779 with a mean of -0.0803 and a standard deviation of 3.1417. In the previous environment, the velocity of an agent rarely surpassed 6m/s however in this episode with the new dynamics, the last 4 steps of the episode shows velocities over 7m/s. The fuzzy rules that correspond to these new states have rarely been fired, if at all; and thus the calculated temporal differences are large. It is important that these rules must adapt for the new states that the change in environment has presented.

By looking at Fig. 4 it is clear that studying the temporal difference during a game can help us determine when the environment has changed in order to start re-learning; but also when we can stop learning too.

5 Proposed Method: Hierarchical Reinforcement Learning

In order to create a switch to turn learning on/off, we have chosen to make a hierarchical learning model where the agents play their game in the lower level and the higher level learns when to turn learning on and off in the lower level based on the temporal differences calculated in the game. An initial idea to determine environmental changes was to use a threshold value, and if the standard deviation of temporal differences passed this value, learning would switch on. However, due to the highly non-linear nature of the problem, we were unable to successfully find any threshold values through trial and error. This led to hierarchical learning where a fuzzy inference system could be used to select when learning can be switched on and off based on the temporal difference statistics of the lower-level policy.

Fig. 8 shows a diagram of the hierarchical learning model. The actual game is played in the lower level, after each episode the standard deviation of temporal differences and the mean of the temporal differences in the game are calculated. The lower level policy is the controller of the agent. The higher level policy is used to determine whether this controller of the agent must be updated to reflect a new environment or not.

These temporal difference statistics are used as the input to the actor and critic of the HLP to determine if the learning must be on for the next episode; these statistics are also as part of the HLP reward. The reward function selected (18) seeks to minimize the standard deviation of temporal differences and mean of the temporal differences in the game. The temporal difference is calculated through the value function seen in (17).

$$\Delta = r_{t+1} + \gamma V_{t+1} - V_t, \tag{17}$$

Another term used for temporal difference is prediction error. The assumption here is that a lower mean and standard deviation of the temporal differences played within a game implies that the prediction error is low, and the game episode is then successful. Note that important hyperparameters such as the discount factor, γ, are given values in Table 1 and Table 2.

The HLP reward is given by,

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

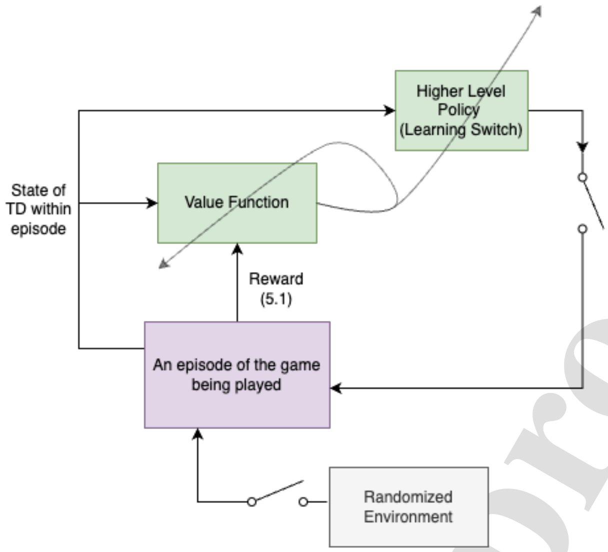


Fig. 8 Block Diagram illustration of a switch designed to learn to adapt policies. The switch learns using an actor-critic learning scheme. Randomized environment indicates that the environment dynamics will suddenly change every 500 episodes for policy learning purposes.

where a and d are values selected by the user. These values are specified in the discussion section as they largely depend on the game and reward structure used. The reward function is structured in a way that gives a higher reward value when μ, the average of the TD within an episode of a game is lower, and when σ, the standard deviation of those TD values is lower. Lower TD statistics generally imply that the prediction error is low because the policy has been successfully learned or adapted.

Fig. 9 shows a diagram illustrating how the HLP is trained. At the start of learning the higher-level switch, the switch is set to 0 which indicates that learning in the lower-level is not taking place during the initial HLP training episode. A switch value of 1 indicates that learning in the lower level is turned on. This on and off learning is to adapt the consequent parameters of the lower-level policy. After the entire lower-level game is played, all temporal differences that were calculated within the game are averaged and the standard deviation is found using (20) and (19). These statistics of the temporal differences are used as inputs to the higher-level learning switch.

$$\sigma = \sqrt{\frac{\sum_{t=0}^{T}(\Delta_t - \mu)^2}{T-1}}. \tag{19}$$

$$\mu = \frac{\sum_{t=0}^{T}\Delta_t}{T}. \tag{20}$$

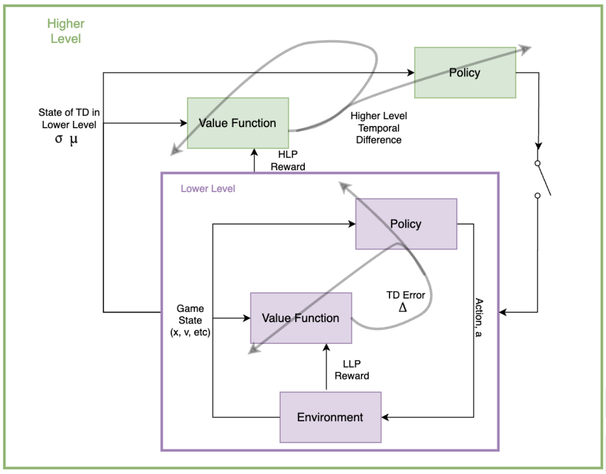


Fig. 9 Higher Level Actor Critic Learning used to switch learning off in the lower-level game actor-critic learning structure.

The reward is calculated by (18) where the constants a and d are selected by the user based on temporal differences seen within the episode of the game played. The output parameters of the rules are updated by (14) for the actor and critic of this higher level.

The mean and standard deviation are used as inputs into the actor to decide if the upcoming game about to be played should have the agent learn or not. A positive output from the HLP represents learning on, and a negative output means that learning should be off. In the continuous hallway game, there is one learning switch for the group of cooperative agents. In the ball balancing task and the guarding a territory game, each agent has their own learning switch. If the learning switch is on, then the agent will add noise/exploration and update the actor and critic fuzzy rule parameter during the episode being played.

To train the HLP, some aspect of the environment is changed every 500 or 1000 episodes. In the cooperative games, 1000 episodes were used and in the competitive game, 500 episodes were used. The number of episodes between environment changes to train the HLP generally will depend on the game and LLPs that are being used. In cooperative games, it may take longer to reach a stable LLP after an environment change due to having to coordinate with other agents in the environment, hence why 1000 episodes are used to adapt rather than 500 for training HLP purposes. The proposed method is shown in Algorithm 1.

An important aspect of the proposed method is that when there is an environment change such as the mass of the agent changing, the corresponding rules that fire in the LLP will be required to be updated. An increased mass of an agent might require a larger force to be output, thus ζ_{LLP} and ω_{LLP} will need to be updated and overwritten. Due to this, the previous environments will be 'forgotten' within the LLP. An important reason for this is that remembering the required policy for every single

Algorithm 1 The proposed algorithm: Learning Switch

Initialize:

Set the hyperparameters which can be found in Table 1.

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 ω and critics ζ 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 signify the learning state of on (positive) or off (negative). 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 with (9).

The actions are taken and the agents move to the new state based on the dynamics given by (1) or (3).

The reward, R_{t+1} for the LLPs are received, with (1) or (4).

Temporal differences are calculated with (13) 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 with (14) and (15).

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. This is done with (18). In some cases, implementing a filter on TD is beneficial.

Update the actors and the critics of the HLPs using (14) and (15).

if Iteration number modulus $500 == 0$ **then**

The parameters of the environment dynamics in (1) or (3) are randomized to give a new environment.

end if

end for

Finalization: Store the policies.

environment change becomes much more computationally expensive along with a much larger memory required. By adapting the policy to constantly match the environment, the amount of memory and computation required is much smaller and more efficient.

6 Results and Discussion

In this section, we apply the proposed learning switch method to the differential games described in section 3. Results of how the HLP performs at adapting the LLP during environment changes are studied. The consequent output rule parameters are studied during the environmental changes as learning adapts the LLP. First, we go through the preliminaries and hyper-parameters of the simulations. Then, we look at the cooperative differential games, and finally we study the competitive differential game. The code scripts of this paper are written in Matlab 2021b and the results are simulated by desktop PC that runs on an Intel Core i5 CPU.

6.1 Preliminaries

During the training of the HLP, the dynamics of environment will change in some way every 500 to 1000 episodes. For example, the mass may suddenly change from 0.1kg to 1.4 kg at training episode number 3000 and stay that way until until episode 3500. During those 500 games, the HLP must learn when it is appropriate to have the LLP adapt its consequent rule parameters based on the temporal difference statistics.

Table 1 Parameters used in training the cooperative policies

	LLP		HLP	
Parameter	Continuous Hallway	Ball Balancing	Continuous Hallway	Ball Balancing
Number of Membership Functions	7	15	15	25
Number of inputs	4	2	2	2
Maximum Training Episodes	50,000	5000	80,000	100,000
Maximum Time in each Episode (seconds)	30	10	N/A	N/A
Actor Learning Rate	0.05	0.05	0.3	0.3
Critic Learning Rate	0.1	0.1	0.5	0.6
Noise	0.9	0.9	0.9	0.9
Discount Factor	0.995	0.9	1	0.5
Reward Weight	$w = 0.998$	$w = 0.8$	$a = 0.25, d = 0.25$	$a = 0.1, d = 0.1$

The algorithm used to train the higher level switch to turn on and off the learning in the lower level is outlined in algorithm 1. This algorithm uses a pre-trained lower level policy. The game's environment will change every 500 episodes. By changing the dynamics of the game, the temporal differences within the game will dramatically change, which should signal the higher level policy to turn on learning in the lower level. An environment change may be abrupt or constantly changing. In both cases, there will be changes in the temporal difference. The change in the TD will trigger the learning to switch on. With 100,000 training episodes, only the environment that the agents play in will only change 200 times. This way there will be 200 peaks of temporal difference changes that the higher level switch can learn from.

Once the HLP is trained, it can be executed as a learning switch in non-stationary environments. Since there are only 2 inputs into the higher-level policy FIS actor, there are not many rules; the computation is quick. After each episode of the game, the lower level temporal difference statistics are calculated. These values are then input into the HLP which determines if the environment has changed and the policy parameters must adapt or not.

For the competitive game, we modeled the pursuit-evasion game of section 3.2 with two players. In the beginning of each episode, the invader and the defender are transferred to an arbitrary location within a bounded location. The invader's initial location is (5,5) and then perturbed by adding a Gaussian noise with mean of 0 and standard deviation of 1. The defender's initial location is (30,30) and then perturbed by adding a Gaussian noise with mean of 0 and standard deviation of 1. The game field is a 50×50 square. The goal location is what makes this game non-stationary: every 1,000 epochs, a new goal location is selected. When we are pre-training the LLP, the goal location is fixed on (10,40). The goal location will change randomly during the HLP training. The agents' speeds are 1.0 $unit/sec$, and the distance between two axles, the parameter L in (5) is set to 1.0 $unit$. The capture radius is equal to 2.0 $unit$s. The other hyper-parameters of the competitive game are shown on Table 2.

Table 2 Parameters used in training the competitive policies

Parameter	LLP	HLP
Number of Membership Functions	5	10
Number of Inputs	4	2
Maximum Training Epsiodes	5,000	20,000
Maximum Time in each Episode (seconds)	100	100
Actor Leaning Rate	0.25	0.1
Critic Learning Rate	0.5	0.2
Noise	1	0.1
Discount Factor	0	0.5
Reward Weight	$W_I = 0.675$, $W_D = 0.45$	N/A

We trained the HLP and LLP system several times with different seed numbers to have several different initial conditions and finally we reported one case in our paper. But here is an interesting fact in our particular case. At the terminal state of each game, a new initial location for the agents will be selected. Thus, the agents will be trained to operate in different locations of the environment. On the other hand, the initial actor's and critic's output parameters at the very first time step are always zero. But when we are in the middle of the game, and the environment changes, the initial output parameters are the same as the output parameters just before the change happens. Thus, in training the LLP, we will have different initial output parameters at each time when the we have a change in the environment.

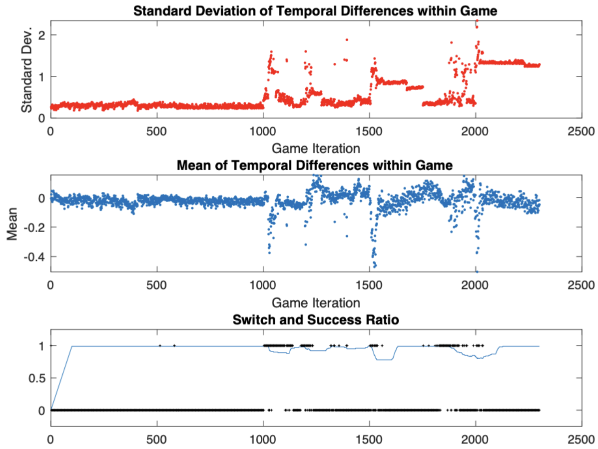


Fig. 10 Second example of HRL results for the continuous hallway game. The top plot shows the mean temporal difference in an episode, the middle plot shows the standard dev. of temporal difference in an episode. And the bottom plot shows the success rate and state of the learning switch (on=1 or off=0) for a given episode.

6.2 Cooperative

6.2.1 Continuous Hallway

Recall that the goal of the two agents in the continuous hallway game is to reach the end of the hallway at the same time. This implies that one agent may need to slow down so that the other can catch up. But what happens when the mass of the agents change from one game to another? By using the hierarchical learning model to learn when the mass of the agent m and friction coefficient of the hallway b changes. Then the HLP will turn on the adaptation of the consequent parameters for the LLP. This will reduce the computational requirements for the agent because adaptation and learning for the LLP will only be done when needed as determined by the HLP. The parameters used for training in the LLP and HLP are found in Table 1. In this game, there is one HLP that turns learning on and off for both agents. The inputs into the HLP are the temporal difference mean and standard deviation of a single agent.

The initial LLP was first trained for 50,000 episodes. 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. The HLP had to learn when it was appropriate to turn learning on and off during these changes. Each episode runs for 30s.

Fig. 10 shows an example of a successfully trained HLP. A successful HLP should both react fast when an environment has changed in some manner, and should adapt the consequent parameters of the policy quickly. Fig. 11 shows the exact environment transformations that occur within Fig. 10 along with how many episodes had learning

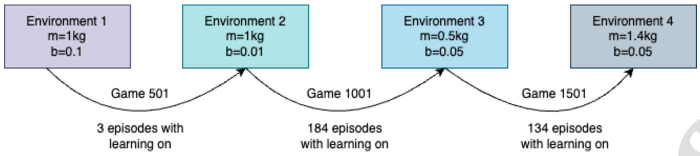


Fig. 11 Environment changes that occur in Fig. 10

on to adapt the lower-level policy. Fig. 10 contains 3 important plots. The top plot contains points that represent the mean of all the temporal differences of a single agent within a single episode played. The middle plot shows points that represent the standard deviation of all the temporal differences for an agent within a single episode played. The bottom plot keeps track of the score and the learning switch. A black point is used at y=1 to indicate that learning was turned on, and at y=0 to indicate that learning is turned off within the LLP. The blue line plots the running score of the last 100 episodes played. For example, a score at 0.5 indicates that only 50 of the last 100 games are successful. Its important to note that in this cooperative differential game, there is only one HLP that controls both agents; it takes the temporal difference statistics from only one single agent.

The example result is shown in Fig. 11. From This initial LLP was trained with $m = 1kg$, and $b = 0.1$ in 1. In this example, the environment changes 3 times, at the 500, 1000 and 1500 episode mark; information is summarized in Fig. 11. A brief look at the results in Fig. 11 shows 3 distinct temporal difference shifts within the plots. At episode 501, the coefficient of friction, b, changes from 0.1 to 0.01. A short delay occurs and at episode 514 the learning switch turns on. There are then 3 episodes total that are played with learning turned on. This shows the robustness of the pre-trained policy.

At game 1001, the mass, m 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 turns learning on for the LLP. We can see how the standard deviation of TD jumps up and the mean of TDs becomes negative. It takes 184 games for the LLP output parameters to adapt to this environment change. The standard deviation of TD settles to approximately 0.3. Comparing the LLP performance prior to the environmental change at game 1001 to the LLP after it converged to its new policy after the environmental change, we see, we see some minor and major changes in the values of the output parameters that were adapted. In total, 285 rule output parameters were adapted for agent Diana and 279 rule output parameters were adapted for agent Sharon.

The largest consequent parameter adaptation in Sharon's LLP went from a value of 2.9390 to 0.9306. More interestingly, agent Diana's largest adaptation occurred with rule 1494 from -0.0628 to 2.0131. Recall that these parameters make up the force that the agents output into their dynamic system. A negative value tends to imply a negative force, or a force in the opposite direction from the finish line. Further analyzing the adaptation that occurs to rule 1494 during this environment change, this rule is made up of 4 triangular membership functions: $r_{1494} = [MF_1, MF_2, MF_3, MF_4]$. Where $MF_1 = [6, 8, 10], MF_2 = [1, 2.5, 4], MF_3 = [-4, 0, 4], MF_4 = [1, 2.5, 4]$. Recall that each membership function corresponds to part of the input state; in this case

MF_1 corresponds to the position of Diana, MF_2 corresponds to Diana's velocity, etc. This means that when an input is within these ranges, rule 1494 will fire. The membership function is made up of 3 values indicated the extreme values and the peak value of the triangle. This implies that Diana must speed up when these rules fire in the new environment and thus an adaptation into the positive force direction occurs to catch up. Analyzing the rules that fire as they adapt shows the functionality and transparency of using fuzzy systems as function approximators for the actor and critic.

Finally, at episode 1501, the environment changes in a greater manner, with the mass increasing to $m = 1.4kg$ and the friction coefficient, b, kept at 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. In Sharon's LLP, the largest adaptation that occurred was rule 1887 from -1.0591 to 2.2580. This adaptation occurred due to the mass increases from 0.5kg to 1.4kg. Here we can analyze a set of rules that fire somewhat frequently near the end of the hallway that contains rule 1887. The rule set is: [1830, 1831, 1837, 1838, 1879, 1880, 1886, 1887, 2173, 2174, 2180, 2181, 2222, 2223, 2229, 2230]. Recall that since there are 4 inputs into the system and triangular membership function are used, $4^2 = 16$ rules will fire. Looking specifically at rule 1887, it is made up of $r_{1887} = [MF_1, MF_2, MF_3, MF_4]$. The triangular membership functions are given as $MF_1 = [12, 14, 16], MF_2 = [1, 2.5, 4], MF_3 = [-4, 0, 4], MF_4 = [1, 2.5, 4]$. This implies that rule 1887 will fire when the inputs are within these ranges, and these outputs required the biggest correction for this rule. For the entire rule set, the output parameters were adapted as shown in Table 3.

Table 3 Rule Set Firing Example in the Continuous Hallway Game

Rule #	Before Change	After Change
1830	2.9435	2.8929
1831	2.9994	3.0000
1837	0.5309	1.5119
1838	-0.1515	2.9561
1879	2.8054	2.8175
1880	2.9989	2.9777
1886	-2.4569	-1.7898
1887	**-1.0604**	**2.2580**
2173	0.9748	0.9264
2174	0.8216	0.8050
2180	-1.1151	-1.4278
2181	-1.8928	-1.4544
2222	0.7557	0.7527
2223	0.7836	0.7996
2229	-2.8640	-2.9564
2230	-2.9393	-2.6849

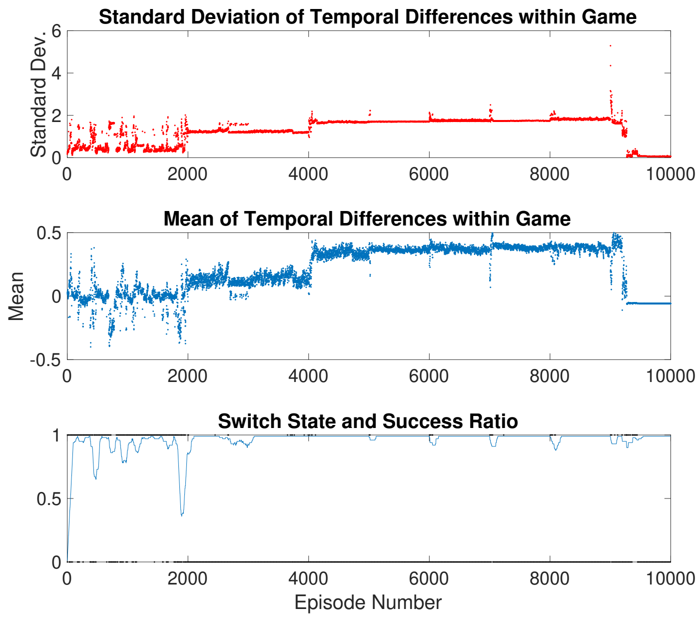


Fig. 12 Example of the higher and lower level policies in action

In Table 3, we see the output parameters from when $m = 0.5$ and $b = 0.05$, and then after the environment changes to $m = 1.4$ and $b = 0.05$. Rule 1838 is the only other rule in this set that changed signs. In this rule set we see a high number of positive values indicating that a force output by the agent is likely to be large and towards the finish line. This change in rules may be due to the increase in mass of the agents, a larger force is required to get the finish line of the hallway.

Another example of a the HLP is shown in Fig. 12. After every 1000 episodes a new mass and friction coefficient are suddenly implemented. This is apparent by looking at the temporal difference plots, at the start of each environment change both the mean and standard deviation vary a lot for several episodes. The standard deviation looks like a streak on the plot; as discussed in Section 4.4 this is often the result of new fuzzy rule output parameters being learned.

It is interesting to note how quickly the LLP adapts once the HLP switches learning on as seen in the bottom plot of Fig. 12. For example, after episode 5000 the environment changes, it took 4 games with the learning on for the success rate to climb back up to 100%. The learning first switched on at episode 5004. After the environment dynamics changed once again at episode 6000, the higher-level policy turned learning on immediately after the first unsuccessful game. This unsuccessful game with the new dynamics had the standard deviation of TDs within the game jump to 2.07. The HLP 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.

This second example which uses the same HLP, shows how quickly both agents' LLP is able to adapt once an environment change occurs. This indicates that the temporal difference is a strong indicator of environmental change.

6.2.2 Balancing A Ball

The balancing ball game described in Section 3 is used as an additional example of the temporal difference learning switch. Recall that in this coordination game, the agents must balance a ball on a table by lifting or lowering the ends of the table.

The hierarchical reinforcement learning method is once again applied to allow the agents to adapt their policies when the environment changes. Environment changes in this game can be change of mass of the ball, change of friction coefficient, and change in length of the table. After some different trials, we found that the initial LLP that was trained was very robust and often did not require any additional learning to adapt when the mass of the ball or the friction coefficient was changed. This meant that much larger changes were required in order to impact the LLP. In order to train the HLP the changes in the environment had to be very large, so 3 changes to the environment were made every 1000 training episodes. The first set of changes were random values between 0 and 1 for the mass and coefficient of friction. The table was set to 3 meter long. At the next environment change, the mass and coefficient of friction are changed to values between 0 and 0.1 with the table being 1 meter long. By going back and forth between these two scenarios we are able to create more extreme cases that require lower-level learning to be on.

The temporal differences seen in this game are much lower due to the reward function. Since the temporal differences are smaller, a and d in the HLP reward function (18) were chosen to be both 0.1. During training of the HLP, 100,000 training episodes were played, since the environment changed every 1000 games, this means there were 100 randomized environments used in the HLP training. Once again, the two inputs into the HLP were the mean of the temporal differences played within the game, and the standard deviation of those temporal differences. In this scenario, both agents have their own switch. Table 1 shows the parameters that were used to train the higher-level policy. The discount factor was chosen arbitrarily but intended to help balance the stability and time horizon aspects of the learning.

Fig. 13 shows the results of a successful HLP from agent Sharon. Both agents produced very similar results so only Sharon's results will be shown. In this plot, there are two disturbances made to the system. The original LLP was trained with a mass of $m = 0.1$, and a coefficient of friction of $b = 0.1$ with the table being 1m long. At episode 1500 the dynamics change to $m = 1kg$, and $b = 0.01$ with the table length changing to 3m. At episode 2500, the mass is decreased to $m = 0.1kg$ and $b = 0.05$ along with the table length decreasing to 1m. Fig. 14 shows a plot where there is no HLP, and the

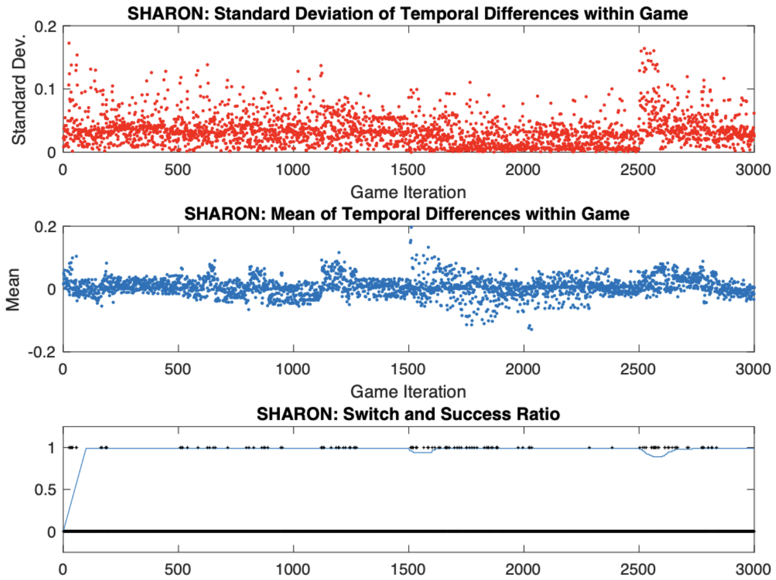


Fig. 13 HLP Results of Balancing A Ball. These plots show how the higher level policy - the learning switch - is successful in recognizing environment changes through the temporal difference and switching on and off the learning accordingly.

learning stays off throughout the environment changes. The information surrounding the exact environment changes that occur in Fig. 13 and 14 is summarized in Fig. 15.

We notice two interesting aspects when comparing Fig. 13 to Fig. 14. First, the lower-level policy is quite robust. The results show that the agents are still successful about half of the time when no learning takes place at all. A successful episode is defined as not dropping the ball off the table for 5 seconds. The second important aspect we see from comparing these plots is that at episode 1500, the standard deviation of temporal differences actually decreases but the mean increases. At episode 2500 the opposite happens. This shows the importance of having both of these statistics as inputs into the HLP.

From episode 1 to 1500, there are 47 episodes with learning. This indicates that the statistics of temporal difference did not satisfy the HLP which rewarded low means and standard deviations. At episode 1500 the environment changed, and 46 episodes following this change had learning switched on. The success rate dipped initially from 100% to a low of 94%. There were 119 episodes with and without learning that passed before regaining the 100% success rate. The first episode that included learning from the HLP occurred 8 episodes after the environment changed. At episode 2500, the environment changed once again. There were 31 episodes that had learning switched

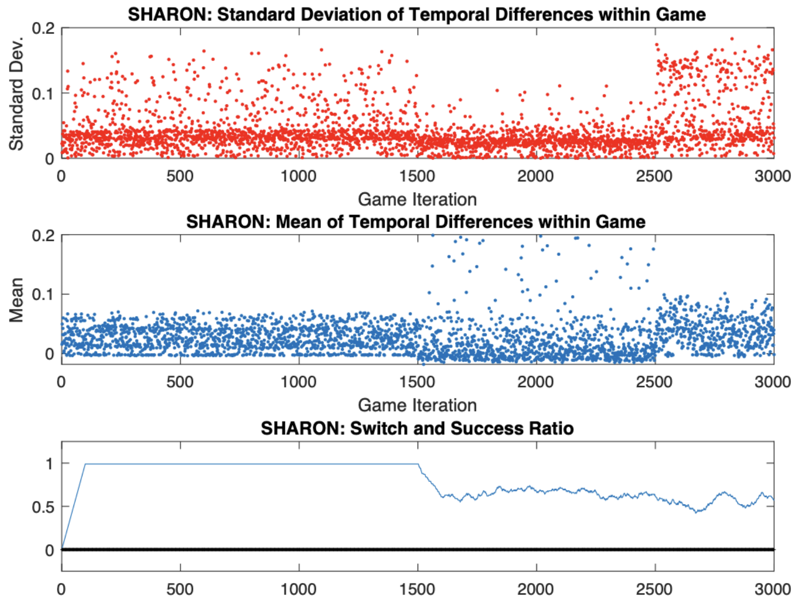


Fig. 14 The impact of the environment changes on the temporal difference statistics and success rate when learning is never switched on.

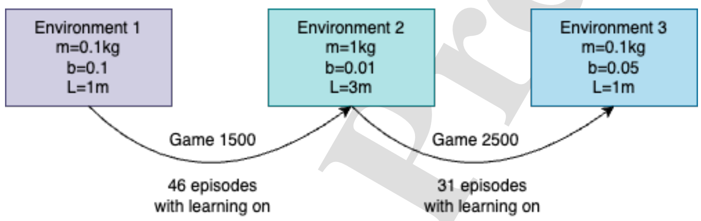


Fig. 15 Diagram showing the environment changes that occur in the ball balancing example in Fig. 13 and 14

on. The lowest success rate recorded was 89%. The first episode that included learning occurred 2 episodes after the environment had changed.

Looking at Sharon's LLP adaptations that took place, the largest rule weight change was rule 157. This rule fired 41,067 times after the environment changed. The original policy had this fuzzy rule weight at 0 and the policy after the first environment change converged this rule to -1. This indicates that a new rule was learned after this environment change. Rule 157 is represented by $r_{157} = [MF_1, MF_2]$ where the triangular membership functions are $MF_1 = [0.75, 1.125, 1.5]$ for the position state and $MF_2 = [-1, -0.5, 0]$ for the velocity state. Two inputs with triangular membership functions has up to 4 rules firing at a given state. Since the initial policy was trained using a table of 1m length where the possible position states went from [-0.5, +0.5], and as such rule 157 would have never fired during the training stage and it is expected rule 157 would have a significant change.

After the second environment change, the largest adaptation in Sharon's LLP was rule 128 with a difference of 1.4406. Rule 128 went from a value of -0.4602 to 0.9804 after the policy had adapted from the environment change. This rule was fired 118,766 after the environment change. Rule 128 is represented by $r_{157} = [MF_1, MF_2]$ where the triangular membership functions are $MF_1 = [0, 0.375, 0.75]$ for position state and $MF_2 = [-0.5, 0, 0.5]$ for the velocity state. Since this rule is found in the middle of the table with slow to near zero velocities, it makes sense that this rule is fired so frequently as the reward maximizes low velocity at the 0m mark of the table.

If we look at a set of rules that fire when the ball is at position +0.1m with a velocity of -0.1m/s we see that different rules are fired. Table 4 shows Sharon's rules before and after the second environment change and Table 5 shows Diana's rules before and after the same environment change. Part of the reason the values are so different is because of the length of the table. Before episode 2500 the table was 3m long and after it was 1m long. Calculating the angle of incline of the table shows that they are somewhat close in values. Before episode 2500, the angle of incline with this set of rules is 4.46 degrees and at episode 3000 the angle is 5.517°. Since the ball becomes ten times lighter while the table shortens, a steeper incline is necessary.

Table 4 Agent Sharon rule firings for a given state

Rule	Rule Firing Strength	Rule Weight Before Episode 2500	Rule Weight at Episode 3000
112	0.13416	-0.9997	-1
113	0.63249	0.3612	-0.633
127	0.04083	-1	-0.9997
128	0.1925	-0.4601	0.9804
Actor Output		-0.0351	-0.3866

Table 5 Agent Diana rule firings for a given state

Rule	Rule Firing Strength,	Rule Weight Before Episode 2500	Rule Weight at Episode 3000
112	0.13416	0.9989	0.9996
113	0.63249	0.239	-0.8438
127	0.04083	0.9962	0.9995
128	0.1925	-0.6594	0.3576
Actor Output		0.1989	-0.29

We see that the temporal difference is an important calculation that can provide information about the learning process and about the environment. When the environment changes and the learned policy is no longer optimal the temporal difference calculated indicates this. A hierarchical reinforcement learning model can be trained and used to dictate when training should continue on a policy. The lower-level policy is trained to play the game while the higher-level policy is trained to indicate when the lower-level policy needs to adapt. In terms of the cooperative multi agent scenarios, we also see that giving each agent their own switch versus having one switch for the group does not make a large difference in outcome

6.3 Competitive

The pre-train process is done for the hyper-parameters in section 6.1. The result is shown in Fig. 16. It is shown that for $W_I = 0.675$ and $W_D = 0.45$ in (6), the capture point is close to the optimal capture point given by the Cartesian oval method [28].

For the competitive game, the LLP is trained for 5,000 iterations. Then, the HLP is trained for 20,000 iterations. At each 1,000 iterations, the goal location changes. Although the goal location changes, the invader's location is always at (5,5) perturbed by adding a Gaussian noise with mean of 0 and variance of 1. In addition, the defender's location is always set to (30,30) perturbed by adding a Gaussian noise with mean of 0 and variance of 1. After 20,000 iterations of HLP training, we conduct a test. The test result is depicted on Fig. 17.

In the test, the LLP output parameters are set to the output parameters of the pre-training phase. The goal location is set to (10,40), and the invader is placed around (5,5), and the defender is placed around (30,30). Fig. 17 (a) shows that in the first 1,000 iterations there is not a significant difference in the TD mean between the proposed method and the non-adaptive LLP. The reason is that the environment is the exact same environment as the initial training. The same result applies for Fig. 17 (b), where there is not a significant difference between the standard deviation of TDs. Fig. 17 (c) shows that in the first 1,000 iterations, the proposed method switched on the training only for five iterations.

At iteration 1,000, the goal location changes. Between iteration 1,000 and 2,000 in Fig. 17 (a) the mean of TD falls by more than -0.05 units. Fig. 17 (c) shows between iteration 1,000 and 2,000 the learning switches on and off many times. As a result, the mean of TD for the proposed method stays around zero. Fig. 17 (b) shows that between iteration 1,000 and 2,000, the standard deviation of both approaches are not different.

Finally, at iteration 2,000, the environment changes again. Fig. 17 (a) shows that the mean of TD of the non-adaptive method jumps up. However, it does not reach zero and has bias around -0.015. On the other hand, as shown in Fig. 17 (c), the learning switches on for a few iterations. As a result, the mean of TD for the proposed method converges to zero. As shown in Fig. 17 (b), there is not a significant difference between the standard deviation of TD.

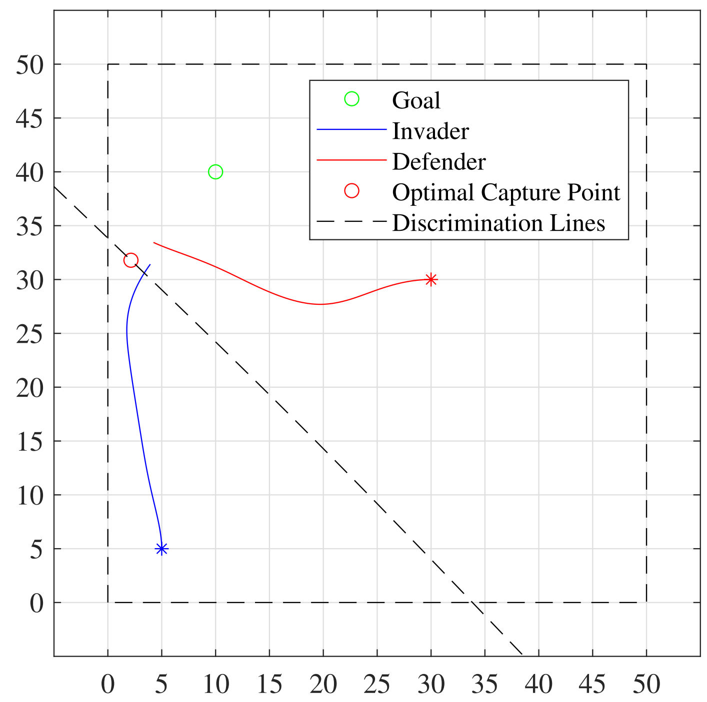


Fig. 16 The trajectory of the invader and the defender after pre-training

7 Conclusion

This paper presented a method to both detect environmental changes in non-stationary reinforcement learning environments, and also to determine when a policy has been properly adapted based on the temporal difference. 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 trained uses a fuzzy logic controller to switch learning on and off in the LLP. The HLP inputs are the temporal difference statistics from the LLP. During the training of the HLP the environment changes every 500 to 1000 episodes. The reward function used to train the HLP learning switch sought to minimize the mean of temporal differences along with the standard deviation. The temporal difference which acts as a prediction error is used as a key indicator to determine if the environment

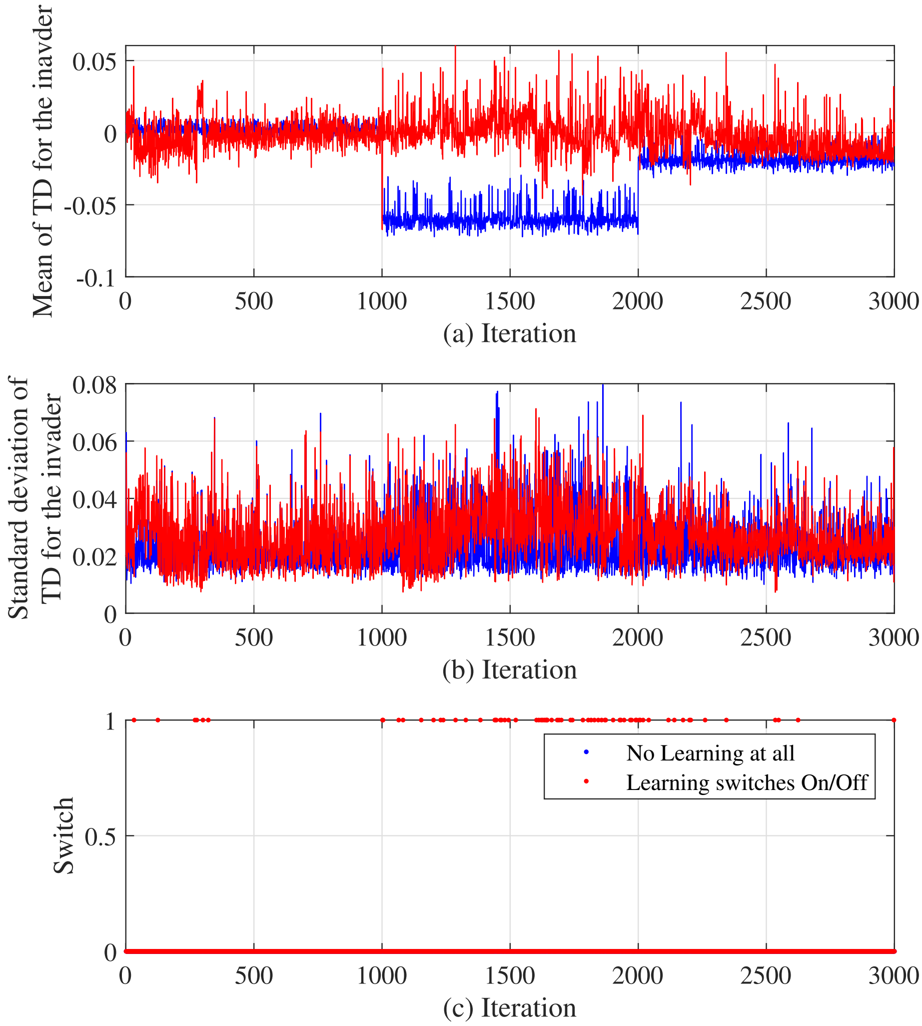


Fig. 17 The performance of the proposed method. (a) The mean of TD for the invader. (b) The standard deviation of TD for the invader. (c) The learning switch of the invader

has changed, and also if the policy has successfully adapted. Calculating the temporal difference for a given state-action is much less computationally expensive than constantly running a learning algorithm in a non-stationary environment.

The applications used to demonstrate this method were multi-agent differential games for both cooperative and competitive games. The results show that this method is successful at adapting the fuzzy rules of a lower-level policy; the HLP is quick to notice an environment change and generally takes minimal episodes to relearn the impacted fuzzy rules.

The contributions are as follows:

- A study of the temporal difference in a reinforcement learning algorithm in non-stationary environments. This paper shows how informative the temporal difference is and how it behaves with both environment changes and further learning. Large changes in the temporal difference statistics often occur when the dynamics of the game shift, and the even larger when there are many new states seen for the first time.
- A hierarchical learning model is developed to learn to adapt a new policy when it recognizes that the environment has changed. More specifically, a method that easily accommodates multi-agent settings. Examples were studied of differential games in both the cooperative and competitive nature. In these examples, the fuzzy consequent parameters being adapted were studied and we saw that adaptation was generally quick; it only took a couple episodes to converge to new parameters. However, this is also dependent on the magnitude of the change to the environment. This proposed method switches learning on and off which saves computational power since learning can be quite costly.
- This paper strengthens the case of using fuzzy systems in the field of reinforcement learning. Fuzzy approximators allow for easy interpretability in machine learning as shown in the discussion of this paper. The analysis of rules during adaptation is simple since each rule corresponds to an observed state.

The proposed method can be called a finite horizon model free approach to reinforcement learning in non-stationary environments. It succeeds at quick adaptation between episodes when the temporal difference statistics demonstrate a shift in values. Since the adaptation is in the form of either turning learning on or off, the computational complexity decreases compared to many other methods. It also succeeds in the interpretability and simplicity of the resultant policy.

References

[1] Weintraub, I.E., Pachter, M., Garcia, E.: An introduction to pursuit-evasion differential games. In: 2020 American Control Conference (ACC), pp. 1049–1066 (2020). IEEE

[2] Isaacs, R.: Differential Games: a Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization. Courier Corporation, Garden City (1999)

[3] Eaton, M., McMillan, M., Tuohy, M.: Pursuit-evasion using evolutionary algorithms in an immersive three-dimensional environment. In: IEEE International Conference on Systems, Man and Cybernetics, vol. 2, pp. 348–353 (2002). IEEE

[4] Asgharnia, A., Schwartz, H.M., Atia, M.: Deception in a multi-agent adversarial game: The game of guarding several territories. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1321–1327 (2020). IEEE

[5] Gregorin, L., Givigi, S.N., Freire, E., Carvalho, E., Molina, L.: Heuristics for the multi-robot worst-case pursuit-evasion problem. IEEE Access **5**, 17552–17566 (2017)

[6] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)

[7] Lau, M., Steffens, M., Mavris, D.: Closed-loop control in active target defense using machine learning. AIAA Scitech 2019 Forum (January) (2019) https://doi.org/10.2514/6.2019-0143

[8] Schwartz, H.: 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 (2019). IEEE

[9] Gu, X., Han, J., Shen, Q., Angelov, P.P.: Autonomous learning for fuzzy systems: a review. Artifical Intelligence Review **56**, 7549–7595 (2023)

[10] Angelov, P., Buswell, R.: Identification of evolving fuzzy rule-based models. IEEE Transactions on Fuzzy Systems **10**, 667–677 (2002)

[11] Rong, H.-J., Sundararajan, N., Huang, G.-B., Saratchandran, P.: Sequential adaptive fuzzy inference system (safis) for nonlinear system identification and prediction. Fuzzy Sets and Systems **57**, 1260–1275 (2006)

[12] Rubio, J.d.J., Bouchachia, A.: Msafis: an evolving fuzzy inference system. Soft Computing **21**, 2357–2366 (2017)

[13] Padakandla, S.: A survey of reinforcement learning algorithms for dynamically varying environments. ACM Computing Surveys **54** (2022)

[14] Yu, J.Y., Mannor, S.: Arbitrarily modulated markov decision processes. In: Proceedings of the 48th IEEE Conference on Decision and Control (2009). IEEE

[15] Dick, T., Gyorgy, A., Szepesvari, C.: Online learning in markov decision processes with changing cost sequences. In: Proceedings of the 31st International Conference on Machine Learning (2014)

[16] Robinson, J.W., Hartemink, A.J., Ghahramani, Z.: Learning non-stationary dynamic bayesian networks. Journal of Machine Learning Research **11**(12) (2010)

[17] Kuznetsov, V., Mohri, M.: Learning theory and algorithms for forecasting non-stationary time series. Advances in neural information processing systems **28** (2015)

[18] Hung, S.-M., Givigi, S.N.: A q-learning approach to flocking with uavs in a stochastic environment. IEEE Transactions on Cybernetics **47**(1), 186–197 (2017) https://doi.org/10.1109/TCYB.2015.2509646

[19] Pickering, L., Cohen, K.: Toward explainable ai—genetic fuzzy systems—a use case. In: Rayz, J., Raskin, V., Dick, S., Kreinovich, V. (eds.) Explainable AI and Other Applications of Fuzzy Techniques, pp. 343–354. Springer, Cham (2022)

[20] Wu, Q., Cheng, S., Li, L., Yang, F., Meng, L.J., Fan, Z.X., Liang, H.W.: A fuzzy-inference-based reinforcement learning method of overtaking decision making for automated vehicles. Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering **236**(1), 75–83 (2022) https://doi.org/10.1177/09544070211018099 https://doi.org/10.1177/09544070211018099

[21] Malik, H., Yadav, A.K.: A novel hybrid approach based on relief algorithm and fuzzy reinforcement learning approach for predicting wind speed. Sustainable Energy Technologies and Assessments **43**, 100920 (2021)

[22] Haighton, R., Asgharnia, A., Schwartz, H., Givigi, S.: Hierarchical reinforcement learning for non-stationary environments. In: 2023 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1421–1428 (2023). IEEE

[23] Wang, T., Wang, J., Zheng, C., Zhang, C.: Learning nearly decomposable value functions via communication minimization. In: International Conference on Learning Representations (ICLR) (2020)

[24] Matignon, L., Laurent, G., Le Fort-Piat, N.: Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2007). IEEE

[25] Asgharnia, A., Schwartz, H., Atia, M.: 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 **18**(6), 1667–1688 (2022)

[26] Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE transactions on systems, man, and cybernetics (1), 116–132 (1985)

[27] Jouffe, L.: 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 (1996). IEEE

[28] Garcia, E.: Cooperative target protection from a superior attacker. Automatica **131**, 109696 (2021)

Rachel Haighton
Rachel Haighton has a B.Eng in Mechanical Engineering from Concordia University in Montreal, QC, Canada, and a M.A.Sc in Electrical and Computer Engineering from Carleton University in Ottawa, ON, Canada. Her research focuses on adaptive and intelligent systems, reinforcement learning, and multiagent systems.

Amirhossein Asgharnia
Amirhossein Asgharnia joined the Department of Systems and Computer Engineering at Carleton University as Post-Doctoral Researcher. He received his Ph.D from Carleton University, ON, Canada in 2023 in Electrical and Computer Engineering. His research focuses on reinforcement learning and multiagent systems.

Howard Schwartz
Howard Schwartz received his B.Eng. degree from McGill University, Montreal, Canada, in June 1981 and his M.Sc. degree and Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, Ma, in 1982 and 1987, respectively. He is currently a Professor in the Department of Systems and Computer Engineering at Carleton University. His research interests include adaptive and intelligent control systems, robotics and process control, system modeling and system identification. His most recent research is in multiagent learning with applications to teams of mobile robots.

Sidney Givigi
Sidney N. Givigi received the Ph.D. degree in electrical and computer engineering from Carleton University, Ottawa, ON, Canada, in 2009. He is currently an Associate Professor with the School of Computing, Queen's University, Kingston, ON, Canada. His current research interests include autonomous systems and robotics.

Declaration of interests

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

☒ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Rachel Haighton reports article publishing charges was provided by Carleton University. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.