**MARMARA UNIVERSITY**

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

# Context Detection and Identification in Multi-Agent Reinforcement Learning on Non-Stationary Environments

EKREM TALHA SELAMET

**MASTER THESIS**

Department of Computer Science and Engineering

**Thesis Supervisor**

Assoc Prof. Borahan Tümer

ISTANBUL, 2022


**MARMARA UNIVERSITY**

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES


# Context Detection and Identification in Multi-Agent Reinforcement Learning on Non-Stationary Environments

EKREM TALHA SELAMET
(524118026)

**MASTER THESIS**
Department of Computer Science and Engineering

**Thesis Supervisor**
Assoc Prof. Borahan Tümer

ISTANBUL, 2022

**MARMARA UNIVERSITY**

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

Ekrem Talha Selamet, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled, "**Context Detection and Identification in Multi-Agent Reinforcement Learning on Non-Stationary Environments**", on 08.07.2022 and has been found to be satisfactory by the jury members.

**Jury Members**

Assoc Prof. Borahan Tümer (Advisor)
Marmara University, Department of Computer Engineering .....

Prof. Dr. Çiğdem EROĞLU ERDEM (Jury Member)
Marmara University, Department of Computer Engineering .....

İnci M. Baytaş (Jury Member)
Department of Computer Engineering, Bogazici University .....

**APPROVAL**

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Ekrem Talha Selamet be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program on ..... (Resolution no: .....).

**Director of the Institute**
**Prof. Dr. Bülent Ekici**

# ACKNOWLEDGEMENT

I would like to thank and express my deepest gratitude towards my advisor, Assoc. Prof. Borahan Tümer, who shared his invaluable knowledge and experience with me throughout the entire thesis process; spared his precious time and demonstrated great patience and interest. It was a great honor to work under his supervision.

Secondly, I wish to express my sincere appreciation to my family; my mother Hülya, my father Kadir, and my sister Nur, for their unconditional support during this process. Without you I could never have reached this current level of success.

Last, but not least, I would like to thank everyone else who has helped me get to this point. Your support will not be forgotten.

**06.2022** **EKREM TALHA SELAMET**

# TABLE OF CONTENTS

# ABSTRACT

## CONTEXT DETECTION AND IDENTIFICATION IN MULTI-AGENT REINFORCEMENT LEARNING ON NON-STATIONARY ENVIRONMENTS

Keywords: Reinforcement Learning, Non-stationary Environment, Context Detection, Multi-Agent, Non-Stationary Sequence

The assumption that the environment the agent learns is stationary has been adopted by many reinforcement learning methods. However, in natural and real-life applications, the environment is non-stationary. One possibility is that non-stationary environments are composed of several stationary components (i.e. context or sub-environment). More than one agent can interact with the environment at the same time, and agents can cause the environment to become non-stationary. The *Reinforcement Learning - Context Detection* method is an approach that enables the agent to learn non-stationary environments without prior knowledge, detects context change points, and identifies contexts. The basis of this approach is single-agent and it has shortcomings for multi-agent learning. In this study a new approach called *Multi-Agent Reinforcement Learning - Context Detection* has been developed that can detect context change points, identify contexts and allow agents to learn the multi-agent non-stationary environment. This approach is based on the reinforcement learning - context detection method; In multi-agent learning, it is more efficient in terms of detecting non-stationarity originating from agents in the environment and detecting context change points. In addition to the context changes caused by the environment dynamics, it also allows detecting the context changes that occur as a result of the changes in the policies of the agents in the environment. In the approach in this study, it has been shown by the experimental results that the agents spend their energy %16 less and are more efficient than the reinforcement learning - context detection method in terms of detecting the context change points more accurately and earlier.

# ÖZET

## ÇOK ETMENLİ PEKIŞTIRMELİ ÖGRENMEDE DEVİNGEN ORTAMLARDA BAĞLAM DEĞİŞİM TESPİTİ VE TANIMLAMA

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Devingen Ortamlar, Bağlam Sezme, Çok Etmenli Öğrenme, Devingen Diziler

Etmenin öğrendiği ortamın durağan olması varsayımı birçok pekiştirmeli öğrenme yöntemi tarafından benimsenmiştir. Fakat, doğal ve gerçek hayat uygulamalarında ortam durağan değildir, devingendir. Devingen ortam, birçok durağan ortamın bir araya gelmesiyle oluşur. Ortam ile aynı anda birden fazla etmen etkileşim kurabilir ve etmenler de ortamın devingen hale gelmesine sebep olabilir. *Pekiştirmeli öğrenme-bağlam sezme* yöntemi, etmenin önsel bir bilgisi olmadan devingen ortamları öğrenmesini, bağlam değişim noktasını sezmesini ve bağlamı tanımlamasını sağlayan yaklaşımdır. Bu yaklaşımın temelinde tek etmen bulunur ve çok etmenli öğrenim için eksiklikleri bulunmaktadır. Bu çalışmada çok etmenli devingen ortamlarda; bağlam değişim noktalarını tespit eden, bağlamları tanımlayabilen ve etmenlerin ortamı öğrenmesine olanak sağlayan *çok etmenli pekiştirmeli öğrenme-bağlam sezme* adında yeni bir yaklaşım geliştirilmiştir. Pekiştirmeli öğrenme - bağlam sezme yöntemini temel alan bu yaklaşım; çok etmenli öğrenmede, ortam üzerindeki etmenlerden kaynaklı devingenliği sezmesi ve bağlam değişim noktasını tespit etmesi yönüyle daha verimlidir. Ortam dinamiklerinden kaynaklı bağlam değişikliklerinin yanı sıra ortamdaki etmenlerin politikalarının değişmesi sonucu oluşan bağlam değişimlerini de sezmesini sağlar. Bu çalışmadaki yaklaşımda, etmenler enerjilerini %16 daha az harcaması ve değişim noktalarını daha doğru ve erken sezmesi açısından pekiştirmeli öğrenme - bağlam sezme yöntemine göre daha verimli olduğu deney sonuçları ile gösterilmiştir.

# LIST OF SYMBOLS

| | |
|---|---|
| $A_s$ | Set of actions in state s |
| $Q^{\pi}$ | Action-value function under a policy |
| $Q^{\star}$ | Action value function under optimum policy |
| $R_m$ | Rewards of a model |
| $S$ | Set of states of the environment |
| $T_a^{\phi}$ | Action trace of an agent $\phi$ |
| $T_m$ | Transition probabilities |
| $T_s^{\phi}$ | State trace of an agent $\phi$ |
| $\Phi$ | The set of agents |
| $\alpha$ | Learning rate |
| $\eta_k$ | Occurrence frequency of heuristic $k$ |
| $\gamma$ | Discount rate |
| $\phi$ | An agent |
| $\pi$ | Policy |
| $a$ | Selected action in a state |
| $e_m^R$ | The quality of reward prediction |
| $r$ | Instant reward |
| $s$ | Current state of an agent |
| $s'$ | Next state of an agent after an interaction with an environment |
| $t$ | Time step identifier |

# LIST OF ACRONYMS

| | |
|---|---|
| BPR | Bayesian Policy Reuse |
| CCDC | Context Change Detection Counts |
| DL | Deep Learning |
| DP | Dynamic Programming |
| DRL | Deep Reinforcement Learning |
| ET | Experience Tuple |
| FOM | First Order Markov |
| HM-MDP | Hidden-Mode Markov Decision Process |
| MARL-CD | Multi-Agent Reinforcement Learning - Context Detection |
| MAS | Multi-Agent Systems |
| MDP | Markov Decision Process |
| MSE | Markovian Switching Environments |
| NS | Non-stationary |
| NSE | Non-stationary Environment |
| NSEs | Non-stationary Environments |
| PS | Prioritized Sweeping |
| RL | Reinforcement Learning |
| RL-CD | Reinforcement Learning - Context Detection |
| SE | Stationary Environment |
| SLWE | Stochastic Learning Based Weak Estimation |
| TD | Temporal Differences |

| | |
|---|---|
| XCS | Extended Learning Classifier |

# LIST OF FIGURES

# LIST OF TABLES

# List of Algorithms

# 1 INTRODUCTION

Humans naturally learn by observing or interacting with their environment. One of the learning methods close to this behavioral learning paradigm is Reinforcement Learning (RL). RL is another sub-branch of machine learning besides supervised and unsupervised sub-branches. RL enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. In other words, the agent learns by interacting with the environment and using the response signals it receives from the environment. The Agent's main goal is to maximize its cumulative reward by reaching some goal state.

The agent continues to interact with the environment until; it reaches the terminal state or reaches a certain action time. While the terminal state may be a state that the agent must reach as a result of its actions, receive rewards at the end and end the episode, it is also the state that causes the agent to die or be punished by existing in a state that should not exist. In RL problems, the agent tries to converge to a near-optimal policy throughout its life. It aims to learn this policy by using its experiences throughout the learning process.

In RL, the environment is usually represented by the Markov Decision Process (MDP). There are two types of environment in terms of their behavior: stationary and non-stationary (NS). A Stationary Environment (SE) is where the rules and dynamics of the environment do not change over time. The rules of the environment are often represented as an MDP model, which consists of all the state transition probabilities and reward distributions. A non-stationary environment (NSE) is where dynamics of the environment, reward and state transitions, can change over time. It is assumed that NSE can be composed of finitely many stationary components (i.e., *sub-environments* or *contexts*), that we call each a context.

Classical RL methods, such as Q-Learning [1], Prioritized Sweeping (PS) [2] are based on the assumption that the environment is stationary. These approaches can find a near-optimal policy on the stationary environment. In other words, they can learn the stationary environment. However, most of the real and natural environments are NS. Classical RL methods fail to learn NSE. After learning the first context, they cannot perceive when the context change occurs and they start to learn the encountered context again. In this process, the agent forgets its past experiences while learning new context. When the agent encounters the context it encountered before, the agent has to learn

it again because it already unlearns this context. So, classical RL approaches cannot learn NSE.

Markovian Switching Environments (MSE) where NSE consist of finitely many stationary contexts. In this work NSE assumptions relies on that. $E_i, i \in [1, \ldots, I]$ where each context $E_i$ is itself a state of a Markov chain [3]. In case the context changes $E_{i_j}$ to $E_{i_{j+1}}$, classical RL methods do not notice the context change and lose their previous experience.

*Reinforcement Learning - Context Detection (RL-CD)* [4] is an approach with context change detection capability in NSE. This approach allows the agent to detect context changes in the environment and the agent creates partial models for each stationary context of NSE. Thus, the agent does not forget his past experiences. When it encounters the same context again, it recognizes the context and uses it if it has a partial model in its memory. Thus, it does not repeat the learning process. It calculates the model quality signal, which indicates how suitable the partial models that the agent has learned before are in the context. If this value is below a certain threshold, it means that the partial models that the agent has learned before are incapable of representing the context in which they are in. That is, it encountered the context for the first time. It creates a new partial mode and begins learning the context. If it has a partial model above this threshold, it means the context it encountered before and it starts using this partial model.

In real-life applications, environments mostly consist of Multi-Agent Systems (MAS). In such environments, multiple agents interact with the environment. During these interactions, in addition to the dynamic variables of the environment, non-stationarity originating from agents occurs. For example, one of the agents may deviate from its usual policy and make choices that will affect other agents. Other agents should be able to realize this change and adapt their policies accordingly. RL-CD is a single-agent method that works well. Therefore, it cannot adapt itself to the actions of other agents in a multi-agent environment. In addition, RL-CD is insufficient to detect non-stationarity that is not related to the dynamics of the NSE but caused by the other agents.

The Multi-Agent Reinforcement Learning - Context Detection (MARL-CD) method developed in this study can perform multi-agent NSE context change point detection and identification. This method is based on the RL-CD method. In MARL-CD, the agent can observe other agents in the environment and creates partial models for each

of the other agents using RL-CD. Thus, the agent detects context changes not only by using its own partial model, but also by using the partial model of other agents it follows. In addition, the agent can make choices by using the partial model of the agents it follows, if it is in the state of an environment that it has not discovered before, or by considering the environment responses as a result of the actions of other agents. It enables agent's to develop policies that will allow it to maximize the sum of rewards as a result of observing other agents. MARL-CD differs from RL-CD in the following aspects: it can detect context change points more accurately and quickly in NSEs with multi-agent settings, identify contexts more accurately, detect policy changes by other agents, and use partial models of other agents during action selection.

It has also been shown by experiments that MARL-CD is more efficient than RL-CD. In the experiments, all agents have the same capability and the environment gives each agent the same response for the same state-action. These environments are NSEs consisting of multiple stationary contexts. Under these circumstances, the following results are observed; the approach is more efficient in terms of maximizing total reward by average 9.58 to -257.75; and minimizing total energy loss by 16% due to the agent's actions.

The main and novel contributions of this study can be summarized as follows:

- Agents create partial models by following competing agents in the environment. Thus, they try to create better policies by analyzing the action and state traces of rival agents.
- Agents detect context change points earlier and more accurately by using partial models of competing agents.
- Agents have the ability to select multi-action (options) in the same step.
- Agents can discover features (heuristics) in certain states of the environment that give reward or punishment in response to the agent. In this way, it can increase the learning speed by making action selections accordingly, and at the same time maximize the cumulative total reward.

The remainder of this study is organized as follows. In the RELATED WORK, previous studies in the literature and the differences of these studies from MARL-CD are mentioned. The theoretical basis of MARL-CD and the details of this approach are explained in the third chapter, METHODOLOGY. EXPERIMENTAL RESULTS presents the

experimental results are interpreted. Different test cases and their results are explained. The last chapter, CONCLUSIONS AND FUTURE WORK, the information obtained as a result of this study is summarized and mentions the improvement aspects of this study.

# 2 RELATED WORK

The changing over time non-stationary (NS) character of real and natural environments has been a compelling factor for Reinforcement Learning (RL) methods. Classical RL methods such as Q-Learning [1], and Prioritized Sweeping (PS) [2] were insufficient in learning environments with NS character. The environments underlying these approaches are based on the assumption that environments are stationary. In these methods, the agent learns the context of the environment. When the environment context changes, it cannot detect the changes and starts learning the new context again. Meanwhile, it forgets its past experiences, and the context it learned earlier. If it encounters the context that was encountered before, the agent cannot notice this and starts to learn the same context again. For these reasons, these methods have shortcomings for learning non-stationary environment (NSE)s. In other words, they are inefficient in terms of waste of time and energy.

## 2.1 Q-Learning

Q-Learning [1] is a model-free, value-based RL algorithm. *Value-based* algorithms updates the value function based on an equation. Most of them uses Bellman equation. The other type of algorithms are *policy-based* which estimates the value function with some policy obtained from the last policy improvement. Q-Learning is an off-policy learner. Off-policy learners learn optimal policy regardless of the agent's actions. $Q^\star$ is the expected value of taking $a$ in state $s$ and then following the optimal policy. For estimating the value of $Q^\star$, Q-learning uses Temporal Differences (TD) [5]. TD is an agent learning from an environment through episodes with no prior knowledge of the environment. The agent keeps a Q table ($Q[S, A]$) which contains quality value of states and actions. In 2.1 Q function, uses the Bellman equation.

$$Q(s,a) = Q(s,a) + \alpha \, [\, R(s,a) + \gamma \; max \; Q'(s',a') \; - \; Q(s,a)] \tag{2.1}$$

## 2.2 Prioritized Sweeping

PS [2] is a model-based RL technique that tries to focus on an agent's limited computational resources to get a good estimate of the environment state values. In every step, this approach uses Dynamic Programming (DP) methods to reevaluate the value of

every state in the environment. PS measures the expected size of the update against its priority metric for each state so that only the most important states are updated. For propagating the values of states, PS is interested to perform actions in the environment. After updating the value of state $s$, it examines all the states where the agent can reach $s$ in one step and assigns a priority-based value to specify the change in their values. A queue is created by prioritizing each state-action pair whose estimated value will change significantly according to the size of the change. The highest priority state-action pair is backed up, and the effect on each of its predecessor pairs is computed. If the effect is higher than a certain threshold, the priority is calculated again and added to the queue. Thus, the effect of changes is efficiently propagated backward until the pairs in the queue are finished.

## 2.3 Hidden-Mode Markov Decision Process

NSEs are defined as having modes, each consisting of stationary parts, namely Hidden-Mode Markov Decision Process (HM-MDP) [6]. Modes consist of different Markov Decision Process (MDP)s. MDPs have the same state and action space with different reward and transition functions. The hidden mode model is a finite set of MDPs. In this method, HM-MDP, can control the transitions of the state by an MDPs. In this approach, states are observable but modes are not. Hidden-mode models can be learned by variation of the Baum-Welch algorithm. The basis of HM-MDP is the assumption that the environment consists of a certain and small number of modes. However, this assumption is often not valid for real and natural environments. This approach falls short in this respect. The NSE definition in MARL-CD is similar to HM-MDP. However, the assumption that the environment consists of a certain and small number of modes is not valid for MARL-CD. Agents learning with MARL-CD can learn an environment consisting of any number of modes.

## 2.4 Reinforcement Learning - Context Detection

Reinforcement Learning - Context Detection (RL-CD) [4] method introduced for learning NSEs. RL-CD enables agents to detect context changes. In this approach, NSEs are composed of several different stationary components (i.e., *sub-environments* or *contexts*). Contexts are represented as state transitions and reward estimates that the agent learns from its interactions with the environment. With this context detection method, the

agent creates partial models of the environment and detects context change points. Thus, the agent recognizes the contexts. The agent learns by creating the partial model of context. Agent calculates *quality signal* which indicates how suitable the current partial model is to the current context. Thus, the agent uses the highest quality model and continues to learn with the partial model appropriate for the context. After the context change, the agent compares the newly encountered context with the partial models it has created with the quality signal, which is formed as a result of the agent policy and defines the compatibility of the policy with the relevant partial model of context. If the quality signal is below a certain threshold, it means that the agent confronted that context the first time. The agent creates a new partial model and starts to learn that context. RL-CD uses a set of models to learn the dynamics of the environment. A partial model $m$ can be thought of as a function of transition probabilities ($T_m$) and rewards ($R_m$).

At each time step, Experience Tuple (ET), $< s, a, s', r >$, of the agent which is the response of the environment received from the environment after the agent interaction. Using ET updates the current partial model by adjusting its model of transition in 2.2.

$$T_m(s, a, s') = T_m(s, a, s') + \Delta T_m(s') \tag{2.2}$$

The moving average of all previous rewards is reward model ($R_m$) and it can be calculated in (2.3).

$$R_m(s, a) = R_m(s, a) + \Delta R_m \tag{2.3}$$

*Confidence value* is the value indicating how many times the agent has attempted an action in a state. The quality of the model is proportional to the confidence value. Using ET for a partial model, the confidence value, $c_m(s, a)$, is calculated as in 2.4. By using this value, the reward quality calculation is as in 2.5.

$$c_m(s, a) = \frac{N_m(s, a)}{M} \tag{2.4}$$

$$\epsilon_m^R = 1 - 2\left(Z_R \left(\Delta R_m\right)^2\right) \tag{2.5}$$

The instant quality of model, $e_m$, is a linear combination of the quality of reward prediction, $e_m^R$, and the quality of transition prediction, $e_m^T$. These values are linearly

interpolated by $\Omega$, which specifies the relative importance of rewards and transitions for the the model's quality. $\Omega$ is a value that can change according to the environment.

$$\epsilon_m^T = 1 - 2\left(Z_T \sum_{k \in S} \Delta T_m \left(\kappa\right)^2\right) \tag{2.6}$$

$$e_m = c_m\left(s, a\right)\left(\Omega\ e_m^R + \left(1 - \Omega\right)\ e_m^T\right) \tag{2.7}$$

$$E_m = E_m + \rho\left(e_m - E_m\right) \tag{2.8}$$

For each model, quality $E_m$ is calculated at each step. The partial model with the highest $E_m$ value best represents the context in which the agent is located, and this model is selected. If none of the models $E_m$ value is below a certain threshold, a new model is created.

RL-CD works well in NSEs with multi-agent settings. But in environment with multi-agent setting, it has shortcomings. It cannot adapt to the actions of other agents. Also, RL-CD cannot detect non-stationarity caused by the other agents in the environment.

## 2.5 Detection of Regime Switching Points in Non-Stationary Sequences Using Stochastic Learning Based Weak Estimation Method

In [7], a new method has been presented that can detect regime switch points in NSEs. This method uses the Stochastic Learning Based Weak Estimation (SLWE) [8] method to estimate the First Order Markov (FOM) probabilities among the tokens used by the system to detect regime switch points. Regime switch points detected when the SLWE estimator unlearns. It learns and detects regime changes by reconverging to a new value that reflects the FOM dependency of the environment output tokens for the new context.

## 2.6 Bayesian Extended Learning Classifier System

In multi-agent environments, single-agent-based learning approaches are lacking, they have several shortcomings: the agent cannot detect non-stationary caused by other

agent's in the environment; the agent cannot realize other agents' policies change; the agent cannot adapt itself to the possible policy change of other agents; it cannot change its policy as a response to other agents' changing policies. In addition to the mentioned shortcomings, the optimal policy that the agent has learned may turn into a sub-optimal policy over time due to the behavior of other agents in the environment. Following the policies of other competing agents in the environment is one of the most efficient methods for comparative multi-agent systems.

In [9], *Bayes-XCS* approach introduced for NS opponents, which is based on the Extended Learning Classifier (XCS) method [10] in Markov games. It introduced the XCS system for multi-agent RL algorithm in Markov games for learning the best answer using the behavior of the opposing agent. By gathering the signals produced by competing agents, corresponding competitor models are created. Policy changes in competing agents can be detected using these models. In other words, opponent models can be used for opponent policy identification and prediction. Also, the Bayesian-XCS method, which is based on the Bayesian Policy Reuse (BPR) method [11], helps the opposing agent to reuse the best response policy by using their models.

## 2.7 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) was first introduced by the *Playing Atari with Deep Reinforcement Learning* [12] study. Generally, at each step DRL agents receive high-dimensional inputs, and make action selection according to deep-neural-network-based policies. This learning method adjust policy to maximize the return with end-to-end method. Basically, in this approach, image, that is, raw pixel information, is used as input. The model in this approach is the convolutional neural network trained as a different variant of Q-Learning. This model outputs a value function as estimating future rewards. Agents learning directly from high-dimensional data such as speech or vision is always challenging for RL. Most successful RL methods that works on this domain have relied on hand-crafted features combined with linear value functions or policy representations. Obviously, the performance of such methods heavily relies on the quality of the feature representation. In recent developments in the field of Deep Learning (DL), they have been successful in extracting high-level features from raw multi-dimensional data in the field such as speech and vision.

In terms of DL, RL has many challenges. First, DL requires a lot of manually hand-

labelled training data. On the other hand, RL must learn algorithms using reward signal, and reward signal is frequently sparse, noisy and delayed. Second, the delay between the action and the resulting reward can be long. This directly affects the action and the reward received negatively compared to matching. Another issue is that most DL algorithm assume the data samples to be independent, but in RL typically encounters sequences of highly correlated states. Furthermore, in RL the data distribution changes as the algorithm learns new behaviours, which can be problematic for DL methods that assume a fixed underlying distribution.

The ultimate aim in this approach is to connect a RL algorithm to a deep neural network which operates directly on RGB images and efficiently process training data by using stochastic gradient updates. In this approach, *experience replay* [13] method utilized where the agent's experiences stored at each time-step in a data-set. It pooled over many episodes into a *replay memory*. After applying experience replay, the agent selects an action and executes it to an $\epsilon$-greedy policy. This algorithm, *deep Q-learning*, uses fixed length representation of histories due to limitation of using histories of arbitrary length as inputs to a neural network can be difficult.

## 2.8 Human-Level Control Through Deep Reinforcement Learning

Deep Q-network [14] approach is a method that makes it possible to learn by taking pixels and game scores as input. This method performed better than previously developed methods using the same algorithm, network architecture, and hyperparameters. This work results in the first artificial agent that can learn to succeed in a variety of challenging tasks introduced that connects the higher-dimensional sensory inputs and actions. In other words, in this study they developed an agent, deep Q-network, which is able to combine RL with deep neural networks to achieve creating a single algorithm, it performs better than previous solutions in various challenging tasks. With recent developments in deep neural networks, more abstract representations of data can be built using several layers of nodes. This makes it possible for artificial neural networks to learn directly from raw sensory data. In this study [14], one specific successful architecture, deep convolutional network, was used. It uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields, inspired by Hubel and Wiesel's work [15].

In this approach tasks are considered as the agent's interactions of an environment

through a sequence of observations, actions and rewards. The main aim of the agent is to select actions in a way that maximizes cumulative future reward. It uses a deep convolutional neural network to approximate the optimal action-value function, (2.9). 2.9 is the maximum sum of rewards $r_t$ discounted by $\gamma$ at each time step $t$, reachable by a policy $\pi = P(a/s)$, after taking an action $a$ and making the observation $s$.

$$Q^*(s,a) = max\mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{2.9}$$

## 2.9 Play Ms. Pac-Man Using an Advanced Reinforcement Learning Agent

In this study [16], popular arcade video game Ms. Pac-Man examined and introduced a new method for dealing with the its large dynamical environment. This approaches main motivation is that speed up the learning process without the necessity of Q-function approximation. Furthermore, this approach focused on the designing of an suitable state space for building an efficient RL agent to the Ms. Pac-Man game domain.

The description of the state space in the Ms. The Pac-Man domain should contain useful information about its position, food, and ghosts. The state space model should define the physical dynamics of the system and establish an efficient relationship between inputs and actions, reflecting the internal characteristics of the system. An ideal state space representation should include these information; the relative position of Pac-Man; the current food knowledge; the modes of nearest ghosts. More specifically, in this approach [16], the state space is constructed as a 10-dimensional feature vector with discrete values. Feature vectors used as a input to learner.

# 3 METHODOLOGY

In this study, based on Reinforcement Learning - Context Detection (RL-CD) [4], a new and novel method called Multi-Agent Reinforcement Learning - Context Detection (MARL-CD) that enables learning in multi-agent non-stationary environments (NSEs) is presented. MARL-CD enables the agent to detect context change points and identify contexts. At the same time, it can detect non-stationarity caused by the dynamics of the environment, as well as that caused by the policy changes of other agents in the environment.

While the agent interacts with the environment, it can also observe other agents in the environment. While the agent creates its partial model using RL-CD, it also creates the partial model with RL-CD using the state and action information of the agents it observes. MARL-CD differs from RL-CD in the following additional aspects: i) the agent creates the partial model of the competing agents in the environment, ii) the agent monitors and analyzes the action and state traces of the competing agents in the environment, compares it with its policy and adapts itself accordingly, iii) agents have the ability to select multi-action (options) in the same step, and iv) agents can discover the features (heuristics) in the environment and adapt the action selections accordingly to accelerate their learning speed.

As mentioned in 2.4 section, RL-CD detects context changes by calculating the *model quality* which is linear combination of the quality of reward prediction and the quality of transition prediction. The quality of the model $m$ updated after each step. If the quality of partial models learned by the agent is below a certain threshold, a new model is created because no model can represent the current context well, and at the same time, it is detected that the context has changed. If the quality of any model that the agent has learned before is higher than the quality of the current model, that model is used. In MARL-CD, calculation of model quality is slightly different than that in RL-CD.

$$\epsilon_m^R = 1 - 2\left(Z_R\left(\Delta R_m\right)^2\right). \tag{3.1}$$

The quality of the reward prediction, $e_m^R$ (3.1), calculated the same as that in RL-CD. The normalization factor, $Z_R$, though, is different. In RL-CD, $Z_R$ is calculated as in 3.2 where $R_{max}$ and $R_{min}$ are the maximum and the minimum values of rewards,

respectively. With the normalization factor (3.1) rescales values from the range $[0, 1]$ to $[+1, -1]$, where $-1$ is the worst prediction quality and $+1$ is the best. In MARL-CD, $Z_R$ calculated as in 3.3, where $R$ is maximum reward. If the environment reward values are not in the range of $[0, 1]$, the normalization factor cannot scale $e_m^R$ to the desired range. So, $Z_R$ is changed as 3.3 to scale within the range of $[+1, -1]$.

$$Z_R = (R_{max} - R_{min})^{-1} \tag{3.2}$$

$$Z_R = (R^2)^{-1} \tag{3.3}$$

In MARL-CD, the agent creates a partial model of the other agents in the environment. Thus, while the agent is detecting context change point, it does not only look at its model but also looks at the model of other agents it follows. It may detect non-stationarity that are not caused by environment dynamics (transitions between contexts), but by policy changes of other agents. If it attempted to detect context change points by only looking at its model, it would be insufficient to detect whether they were caused by other competing agents in the environment. In MARL-CD, the agent can distinguish the cause of the change in the environment by following the competing agents in the environment and adapt its policy accordingly while continuing to use the correct partial model. Also, context changes in the environment can sometimes be caused by very small changes and may be difficult to detect as this change has little effect on model quality, $E_M$ (2.8). In such cases, making decisions by looking at more than one partial model allows these small changes to be detected.

An agent $\phi$ may visit for the first time a region in an environment that $\phi$ has never visited before. Therefore, there is no historical information on $\phi$'s partial model. In this case, $\phi$ has to choose an action, $a$, randomly (since no information is available to $\phi$). However, $\phi$ may benefit from the past experience of another agent, in case one exists with relevant experience at that specific region or state, in particular, by checking the partial models of the other agents $\phi$ follows. If the partial model of any agent contains such information about that state, $\phi$ can make a better choice using this information $\phi$ found at this other agent's partial model. Thus, $\phi$ may maximize the total cumulative reward. Even without making explorative choices, $\phi$ captures information about a state of the environment totally unknown to $\phi$ so far.

By following other agents in the agent environment, an agent monitors their action and state traces and maps them to the relevant partial models. In this way, the agent has information about in which context the opponent agent follows which policies. $T_a^\phi$ and $T_s^\phi$ are the action and state trace of agent $\phi_i$ respectively. The agent monitors and keep $T_a^\phi$ and $T_s^\phi$ for every $(\phi_1, \phi_2, .., \phi_{i-1}, \phi_{i+1}, ..., \phi_I) \in \Phi$ where $\Phi = \{\phi_i\}_{i=1}^I$. The agent can compare its policy by using $T_a^\phi$ and $T_s^\phi$ to ensure that its policy is the optimum. So, the agent can create more successful policies. Also, if the agent detects a context change on its model, it can determine whether the rival agent has changed its policy by analyzing state and action traces. It can choose a partial model accordingly and adjust the action selection policy accordingly.

In MARL-CD, agents can select multi-actions (*options*) which enables an agent to select multiple actions in the same time window. An *option* [17] is a sequence of basic actions, specified to represent a data pattern over time, a concept called *temporal abstraction*. Options are defined by a triplet composed of a policy, a termination condition, and an initiation set. $\pi : S \times A \to [0, 1]$, $\beta : S^+ \to [0, 1]$, and $I \subseteq S$ denote, in respective order, the option's policy, termination condition, and initiation set. An option $< I, \pi, \beta >$ is available at state $s_t$ if and only if $s_t \in I$. Actions are selected according to $\pi$ until the option terminates according to $\beta$ if the option is taken. In a *Markov option*, first the next action is selected according to the probability distribution $\pi(s_t, \cdot)$. After the action is executed, the agent makes a transition to state $s_{t+1}$. In $s_{t+1}$ option may be terminated with probability $\beta(s_{t+1})$ or continue. In case not terminated, $a_{t+1}$ is selected according to the policy $\pi(s_{t+1}, \cdot)$ and continues this way.

In MARL-CD, an agent can make action selection choices using options. Thus, the agent has the ability to change its speed with the options it chooses depending on the state it is in. Since the agent chooses more actions in the same step size, it will move more, so it consumes more energy than when choosing a single action. If the agent receives a reward proportional to its remaining energy at the end of the episode, the reward it will receive will be reduced because the energy it will spend will be more when it takes the option. Thus, agents can learn the optimum balance over time whether they choose options or not. Besides, options can also be used to confuse other agents in the environment. In some episodes by using options, the agent can reach the goal state faster and cause other agents to be punished, and force them to change policy. If the agent did not follow other agents in the environment, the optimal policy that it learned may become sub-optimal as a result of the different behaviors of other agents; hence,

it should adapt to such situations accordingly. Since the agent will interact with the environment more in the same step size, it also increases the learning speed.

Some states of the environment may have occasional *features (heuristics)* (e.g., tokens aka. "food" that intermittently appears in Pac-Man and brings in points/awards once taken) that allow an agent to reach the goal state or receive more instant rewards. Agents can receive more rewards and maximize cumulative rewards by visiting states with this feature. While the agent makes action selection in MARL-CD, it checks whether there is a heuristic in the past of that state in the partial model. If it has received more instant rewards in that state-action pair in the past, or if there is a heuristic in the states that are likely to pass as a result of the actions that it can choose from the state it is in, it takes this into account when choosing an action. The agent learns heuristics over time as it builds the partial model. It makes choices using these heuristics and maximizes the total cumulative reward while reaching the goal state faster.

To explain in more detail how heuristics are learned in MARL-CD, in certain states of the environment, the agent receives a response (i.e., reward or punishment) based on a certain probability distribution originating from heuristics. The reward received in this state can be defined as heuristic. With the help of the agent's partial model, the agent learns over time such typical behaviors of the environment within relevant states. In RL-CD, the quality of the reward prediction, $e_m^R$ (3.1), is affected by heuristics. In the state with a heuristic, the deviation from the reward estimate will be higher, as the agent will receive instant reward or punishment. As a result, $e_m^R$ converges to -1. A decrease in the quality of the reward prediction causes a decrease in the quality of the model, $E_m$. As a result of the convergence of the quality of reward prediction ($e_m^R$) to -1 due to the typical characteristic of the state, context change signal produced but the context did not change. Due to the decrease in $E_m$, though, the agent perceives a false context change. RL-CD cannot handle such situations and detects false-positive context change point.

$$\Delta R_m = \frac{r - R_m(s,a)}{N_m(s,a) + 1} \tag{3.4}$$

In MARL-CD, an agent $i$ keeps a history of reward estimates as a list of two-tuples $\{(r_k, \eta_k)\}_{k=1}^{K_j}$ for each heuristic $j$ where $r_k$, $\eta_k$ and $K_j$ denote, in respective order, the instant reward received (the instant reward values are sharp; hence we have a finite number of real values), its occurrence frequency and the upper limit of the number

of possible rewards for heuristic $j$. It keeps a memory of these two-tuples for each state-action pair. While calculating $e_m^R$, the agent considers the occurrence frequency of the current reward, $r$, received as a response of the environment. If $r$ is in the list of heuristic tuples and is above a certain occurrence frequency threshold (i.e., this is a heuristic, $r = r_k$), the tuple value $r_k$ is taken into account when calculating the $\Delta R_m$ instead of the reward estimate value itself. In a state with heuristic while calculating $\Delta R_m$, when the value of $r$ is deviated from the reward estimate, the $\Delta R_m$, (3.4), become negative value and $e_m^R$ converges to -1. Thus, when calculating $\Delta R_m$, only the reward received with the heuristic effect is included in the calculation. If the reward received is not included in the list of heuristic tuples or is less than the threshold, the reward estimate value is used in the $\Delta R_m$ account. In this way, the $e_m^R$ value is calculated more accurately. Even if the context of the environment does not change, some environment states may have different reward characteristics depending on some dynamics within the same context.

After proven that using heuristics improves context detection accuracy and action selection policy for increasing total cumulative reward, we employed a more general method to categorize instant rewards of arbitrary similarity into the same heuristic. By using the ART2-a [18] approach each heuristic $j$ is learned as a category $k$ represented by a value $r_k$. ART2-a is an extension of ART2 [19] and simple computational system that models the essential dynamics of the ART2. In other words, ART2-a is an unsupervised clustering method for multi-dimensional input patterns. In the context of this study, an input pattern is an instant reward received.

In MARL-CD, ART2-a is used to learn the characteristics of the environment at a state by grouping the instant rewards received by the agent. Thus, an instant reward, $r$ received by the agent may be determined whether or not to categorize into one of the learned clusters (i.e, heuristics). If it is not a heuristic, then a reward estimate is used in the $e_m^R$ calculation. To put it in another way, if the instant reward, $r$, does not qualify into a certain category $k$, it is not a heuristic and the reward estimate is used in the $\Delta R_m$ calculation. If it does, it is a heuristic and the central value of that category $k$ is used for the $\Delta R_m$ calculation.

Algorithm 1 is the formulated version of MARL-CD. Each agent in the environment follows these steps during learning.

As MARL-CD is compared with RL-CD in the context of the applications mentioned

above; experiments have shown that MARL-CD is more efficient than RL-CD within a multi-agent non-stationary environment (NSE) setup in terms of energy use, context identification and detection of context changes.

**Algorithm 1** MARL-CD algorithm

**Let** $\Phi = \{e_i\}_{i=1}^{I}$ be the set of $\Phi$ agents
**Let** $m_k^i$ be the $k$th model of the $i$th agent
**Let** $m_{cur}^i$ be the current model of the $i$th agent
**Let** $T_a^i$ be action trace of $i$th agent
**Let** $T_s^i$ be state trace of $i$th agent
creates and initializes own partial model $m_{cur}^{self}$
creates and initializes the list of other agents' partial models $(m_{cur}^1, m_{cur}^2 \cdots, m_{cur}^{i-1}, m_{cur}^{i+1}, \cdots, m_{cur}^I)$
**while** all agents do not reach the terminal state **do**
    **for** all $e_i \in \Phi$ **do**
        analyzes $m_{cur}^{self}$ and $m_{cur}^i$
        analyzes $T_a^i$ and $T_s^i$
    **while** the agent is do not reach terminal state **do**
        executes step
        updates $m_{cur}^{self}$ parameter via RL-CD
        updates heuristic information for state $s$
        **for** all $e_i \in \Phi$ **do**
            observe action and state of $e_i$
            updates $m_{cur}^i$ using observations via RL-CD

# 4 EXPERIMENTAL RESULTS

The Multi-Agent Reinforcement Learning - Context Detection (MARL-CD) results were also confirmed by experiments. Experiments were performed under the same conditions for the two approaches (MARL-CD and Reinforcement Learning - Context Detection (RL-CD)). First, agents learned with RL-CD, while in the second setting, they learned with MARL-CD. It was carried out in two different experimental environments: Ball-Catching World and Pac-Man, which is a more complex environment compared to this environment. Experiment results were compared.

## 4.1 Experiments Setup

Ball-catching environment [4] is one of the environments where RL-CD is tested. This environment was used to make performance comparisons of algorithms more accurate. In addition, experiments were carried out in the Pac-Man [20] environment, which has a more difficult grid structure and a more difficult goal state than the ball-catching environment. Thus, the performances of the algorithms were also tested in different and challenging conditions.

During the experiments, different model parameters (i.e., learning rate, discount rate, omega) were studied and best parameter values found during our studies. Best parameters were used in both approaches. Total cumulative rewards, context detection accuracy, game wins, and energy levels are compared for each method. Reinforcement Learning (RL) algorithms aim to maximize the total cumulative reward, so this parameter was compared in the experiments. One of the most important criteria for methods working in non-stationary environment is context detection accuracy. The correct detection of the context change points of the environment is one of the factors that directly affect the learning speed and accuracy of the agent. Energy level is a parameter that shows how much energy agents spend throughout their lives. In other words, it shows how much energy the agents save. From this parameter, it can be deduced how fast agents can learn the environment. If the agent is learning fast, its energy will decrease less.

In this study, when comparing MARL-CD as the baseline, RL-CD is used. The fact that other approaches were not tested in the test environments used made it impossible to compare with other methods while creating a benchmark. Some methods based on Deep Reinforcement Learning (DRL) uses Pac-Man environment for testing. These approaches use image as a input not state. Because of this basic concept difference

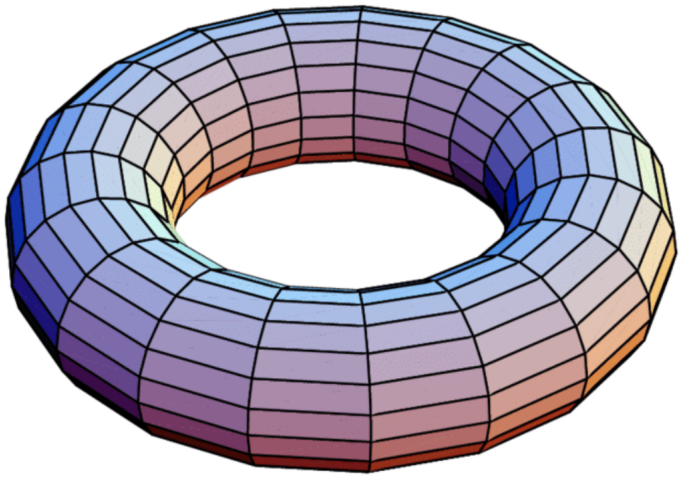

**Figure 4.1:** Representative image of the ball-catching world

when modeling the environment (one of them uses image recognition other one uses basic state information) basing other methods will not be consistent for performance comparison. Also, DRL and RL has some differences in terms of using neural networks. Due to these reasons RL-CD used for performance comparison.

## 4.2 Ball-Catching

Ball-catching world, in which the first experimental environment RL-CD [4] was implemented, was chosen. First validation scenario consists in a non-stationary environment (NSE) built as follows:

- The environment consists of 15x15 toroidal discrete grid cells;
- The agent is a cat and the goal state is catching the moving ball;
- The ball starts its movement in a random state of the environment;
- The action set of ball of the ball is as follows: **i)** *move left*, **ii)** *move right*, **iii)** *move up* and **iv)** *move down*;
- Agent's action set consists of five actions, those of the ball and an additional action *stay*;
- All agents start in the same state initially.

A representative ball-catching world can be seen in Figure 4.1. This environment's

non-stationary (NS) character is due to the change in the direction of movement of the ball over time. While the ball moves in a certain direction over time, it starts to move in a different direction and changes its behavior. For example, as the ball is moving to the right, the context changes if the ball starts to move to the left. In the experiments, the environment context changes every 1000 episodes. Agents are not directly affected by the results of each other's action choices. In other words, as a result of the selection of one agent, the state of the other agent does not change, or the action selection of an agent does not directly affect the action selections of the other. The first agent reaching the goal state, that is, the first to catch the cat gets the determined reward. The losing agents are punished with the same amount. If agents reach the goal state at the same time, they all receive the same amount of rewards. Two competing agents, Agent 1 and Agent 2, were used in these experiments.

**Table 2:** Result of RL-CD Game Wins in Ball-Catching Environment

| Context | Agent 1 | Agent 2 |
|---|---|---|
| 0 | 917 | 721 |
| 2 | 566 | 438 |
| 0 | 981 | 21 |
| 1 | 990 | 89 |
| 0 | 714 | 287 |
| 2 | 697 | 314 |
| 1 | 1000 | 2 |

In the experiments, average game win counts by contexts for RL-CD are shown in Table 2. When we look at the game-winning rates of the agents per context, we see that in some contexts one agent dominates the other. Game winning count is more than the other agent per context. Further the agent who learns the context earlier is superior to the other. An agent's dominance over another indicates that the dominating agent can learn the context, while the other agent cannot. Agent 1 learned faster than agent 2 because of that agent 1 has more game wins. In RL-CD, the agent cannot adapt itself to other agents and every agent in the environment may not be able to learn the context they are in.

Average Context Change Detection Counts (CCDC) per context for agents learning with RL-CD are illustrated in Table 3. Ideally, agents should detect the change point once per context transition. High CCDC show that agents learning with RL-CD are insufficient in detecting context change points. Even if the environment's context does

**Table 3:** Result of RL-CD Context Change Detection Count in Ball-Catching Environment

| Context | Agent 1 | Agent 2 |
|---|---|---|
| 0 | 1 | 1 |
| 2 | 1111 | 2521 |
| 0 | 755 | 37 |
| 1 | 2 | 4 |
| 0 | 1475 | 33 |
| 2 | 3 | 48 |
| 1 | 2 | 1 |

not change, the agent misinterprets the rival agent's possible policy change as a context change. This makes it difficult for the agent to learn the context it is in, and it also reduces the quality of partial models because it uses partial models of the wrong context.

**Table 4:** Result of MARL-CD Game Wins in Ball-Catching Environment

| Context | Agent 1 | Agent 2 |
|---|---|---|
| 0 | 938 | 934 |
| 2 | 539 | 472 |
| 0 | 813 | 641 |
| 1 | 621 | 395 |
| 0 | 998 | 1000 |
| 2 | 692 | 543 |
| 1 | 499 | 802 |

In the experiments, average game win counts by contexts for MARL-CD are shown in Table 4. Agents learning with MARL-CD have approximately equal game win rates per context. Neither agent has dominated the other. This shows that both agents can learn contexts. Unlike RL-CD, in MARL-CD agents have adapted to each other's policies.

The CCDC numbers are shown in Table 5 for agents learning via MARL-CD. Agents detected context change points much more accurately than RL-CD. The low number of CCDC (i.e., it is one for all contexts but context 2) manifests this fact. In the first context change, there are some false-positive signals for context change detection. This is because the changing contexts are very close to each other. The context change in the ball-catching environment is due to the change in the ball's movements. When transitioning from 0th context to 2nd context for the first time, agents cannot detect it directly. Since the dynamics of the two contexts are very similar, the agents had to learn

**Table 5:** Result of MARL-CD Context Change Detection Count in Ball-Catching Environment

| Context | Agent 1 | Agent 2 |
|---|---|---|
| 0 | 1 | 1 |
| 2 | 16 | 7 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 2 | 1 | 1 |
| 1 | 1 | 1 |

by continuing their movements in order to decide on the right context. Unlike RL-CD in MARL-CD, the correct partial models are used in the right contexts. Thus, partial models better represent the relevant contexts. In addition, the use of correct partial models increases the learning speed of agents and reduces the energy they consume. In addition, the choices made by the agents were made in a way that maximizes the total cumulative reward.

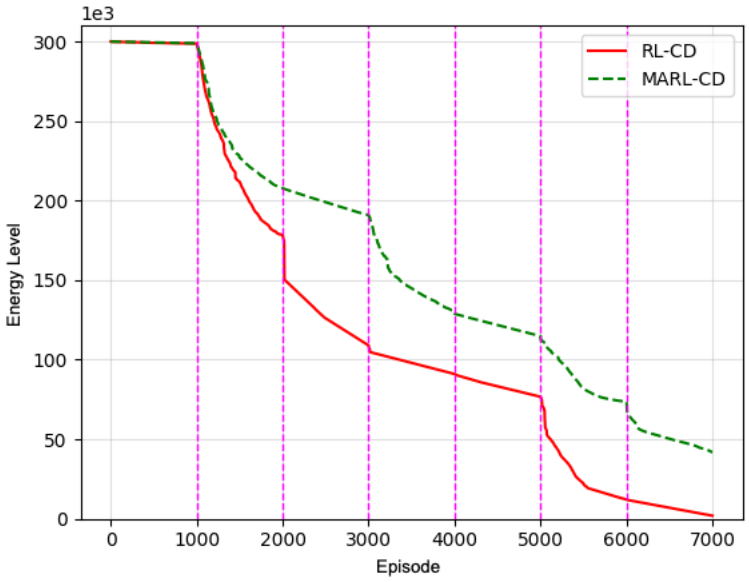


**Figure 4.2:** Graph shows the change in energy levels of agents over time after starting at the same energy level

When we look at the amount of energy consumed by agents per context, we observe in Figure 4.2 that agents learning with MARL-CD consume 16% less energy on average. The vertical dashed lines on the figure show the context change points. In terms of

energy, MARL-CD is more efficient; it is effective to learn faster and to decide on the correct partial model faster. In addition, the ability of agents to take options in MARL-CD also shortens the learning time.

## 4.3 Pac-Man

RL-CD and MARL-CD have also been tested on Pac-Man [20], which is a more complex environment than the ball-catching environment. It is NSE in Pac-Man and has the following setups:

- The environment consists of 18x10 cells;
- The agent is Pac-Man and the goal state is eating all food pieces in the environment;
- If ghost and Pac-Man are in the same state, the ghost will kill Pac-Man and the current episode will be over for that agent;
- Agents are penalized at equal rates when they die as they are rewarded when they attain the goal state.
- For eating a small piece, the agent receives a reward of 10 units while for a large one 100 units and ghosts will be in scared mode;
- Ghosts cannot kill agents while in scared mode and can be eaten by agents;
- If the agent eats the ghost, he gets 200 bonus rewards, and the ghost returns to the start state and starts its life again.
- The set of actions is the same for both the agent and the ghost. The action set is as follows: **i)** *move left*, **ii)** *move right*, **iii)** *move up*, **iv)** *move down* and **v)** *stay*.

The NS character of the environment is based on the ghost's change of movement pattern over time. If agents reach the goal state at the same time, they both get a reward, as in ball-catching. When one agent reaches the goal state before the other, the reaching agent gets the reward and the other gets the same amount of punishment. If an agent dies, it is punished while the other agent can continue the episode. If two agents are caught by the ghost and die before they reach the goal state, that episode will not have a winner and both agents will be punished. The food one agent eats does not affect the other. Figure 4.3 is a representation of the environment. The experiments were repeated 2300 times and the average results were evaluated. During the experiments, the number of episodes passed for the context change was also observed for different

values.

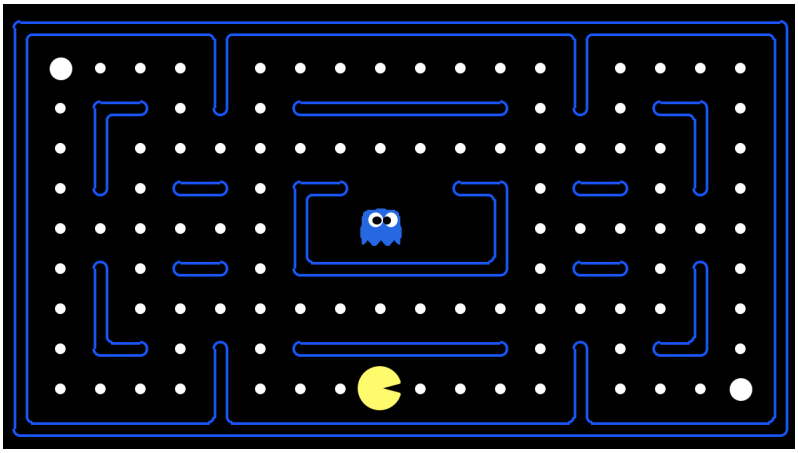

**Figure 4.3:** Representative image of Pac-Man

Table 6 shows CCDC and whether agents use the correct partial models for RL-CD agents. As mentioned before, agents must detect one context change point when the context changes. However, looking at the results in Table 6, agents learning with RL-CD were insufficient to detect context change points and detected many false-positive context change points. In addition, they did not use the context-appropriate partial model. Although there are context changes that it detected, it does not change the partial model it uses in the same context. In other words, it used the same partial model in all contexts and could not decide on the correct partial model as a result of the change points it detected.

**Table 6:** Result of RL-CD Context Change Detection Count in Pac-Man environment

| | CCDC | | |
|---|---|---|---|
| Context | Agent 1 | Agent 2 | Using Correct Partial Model |
| 1 | 0 | 0 | Yes |
| 2 | 19 | 4 | No |
| 3 | 3 | 43 | No |
| 4 | 126 | 213 | No |
| 2 | 11 | 36 | No |
| 4 | 0 | 6 | No |
| 3 | 54 | 23 | No |

In Table 7, the game win counts of the agents and game win rates shown for the agents learned with RL-CD and context changes once in 250 episodes. Since the agents do not

**Table 7:** Result of RL-CD Game Wins in Pac-Man environment

| | Game Win Count | | Game Win Rate | |
|---|---|---|---|---|
| Context | Agent 1 | Agent 2 | Agent 1 | Agent 2 |
| 1 | 23 | 27 | 9.2% | 10.8% |
| 2 | 28 | 19 | 11.2% | 7.6% |
| 3 | 33 | 23 | 13.2% | 9.2% |
| 4 | 17 | 14 | 6.8% | 5.6% |
| 2 | 35 | 22 | 14.0% | 8.8% |
| 4 | 19 | 31 | 7.6% | 12.4% |
| 3 | 27 | 33 | 10.8% | 13.2% |
| Average | 26 | 24.14 | 10.40% | 9.66% |

use partial models suitable for the context, the game win rates did not increase when they encountered the same context again. If they could learn contexts correctly, game win rates would be expected to be higher as the agent would learn much faster when they encounter the same context again.

**Table 8:** Result of MARL-CD Context Change Detection Count in Pac-Man environment

| | CCDC | | |
|---|---|---|---|
| Context | Agent 1 | Agent 2 | Using Correct Partial Model |
| 1 | 0 | 0 | Yes |
| 2 | 3 | 4 | Yes |
| 3 | 5 | 7 | Yes |
| 4 | 4 | 2 | Yes |
| 2 | 3 | 5 | Yes |
| 4 | 21 | 2 | Yes |
| 3 | 3 | 11 | Yes |

Results of MARL-CD are illustrated in Table 8 under the same conditions as those of RL-CD for CCDC and using correct partial model data given. Agents detected context change points correctly and used the relevant partial model for each context. Game win counts and game win rates of agents which learn with MARL-CD are given in Table 9. As can be seen in Table 9, the game win rate is higher when the agent switches back to the context it has encountered before. Contrary to RL-CD, the learning speed is higher because the agent uses a context-appropriate partial model. As a result, the game win rate has been high. The ability of agents learning with MARL-CD to discover heuristics on the environment also affects the high game win rates. Agents learn heuristics over

the partial model over time and determine their action selection policies accordingly. In the Pac-Man environment, foods are heuristics. Thus, while selecting the action, MARL-CD agents select the state-action pairs where the food pieces are located to maximize their cumulative reward and reach the goal state faster.

**Table 9:** Result of MARL-CD Game Wins in Pac-Man environment

| | Game Win Count | | Game Win Rate | |
|---|---|---|---|---|
| Context | Agent 1 | Agent 2 | Agent 1 | Agent 2 |
| 1 | 90 | 89 | 36.0% | 35.6% |
| 2 | 78 | 82 | 31.2% | 32.8% |
| 3 | 110 | 109 | 44.0% | 43.6% |
| 4 | 73 | 68 | 29.2% | 27.2% |
| 2 | 132 | 139 | 52.8% | 55.6% |
| 4 | 99 | 103 | 39.6% | 41.2% |
| 3 | 170 | 183 | 68.0% | 73.2% |
| Average | 107.42 | 110.43 | 42.97% | 44.17% |

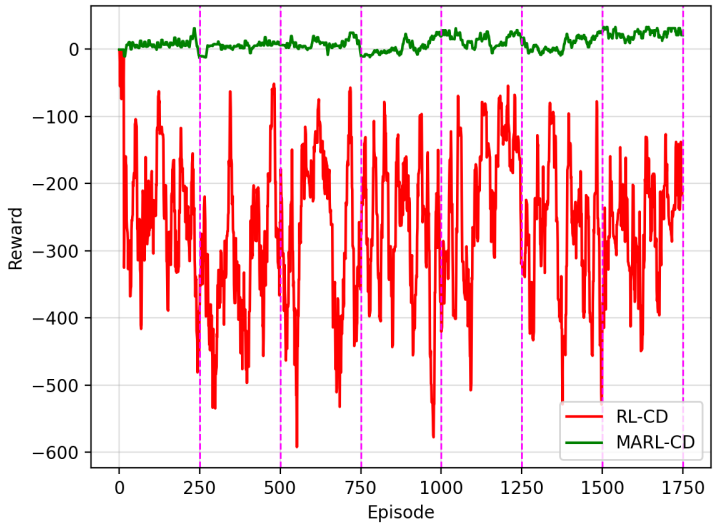


**Figure 4.4:** Average Cumulative Reward Graph of Algorithms

Another metric used to assess and compare performance among RL-CD and MARL-CD in the experiments is the average cumulative reward [21]. Episode cumulative reward is total reward that agent earn in a episode. An episode's average reward was calculated by dividing the episode cumulative reward by the number of steps in that episode. In addition to this calculation, the graphic was smoothened up by taking the moving

average with the window size of ten episodes. As seen in Figure 4.4, the average reward per episode of MARL-CD is higher than that of RL-CD. In other words, MARL-CD agents maximized their rewards at the same time and received higher rewards. While the average reward per episode is -257.75 for RL-CD during the whole learning process, it is 9.58 for MARL-CD. The fact that Pac-Man agents who learned with MARL-CD died less, won more episodes and took fewer steps revealed this difference. It is also seen in Figure Figure 4.4, taking into account the average cumulative reward that MARL-CD converges. When agents policies converge the optimal, the average cumulative reward converges to a certain value.

# 5 CONCLUSIONS AND FUTURE WORK

In this study, a new and novel method has been introduced for multi-agent reinforcement learning operating on non-stationary environments (NSEs). With this method, context change points can be detected, while contexts can be identified. Although it is an approach based on Reinforcement Learning - Context Detection (RL-CD); creating partial models by following other competing agents in a multi-agent environment Multi-Agent Reinforcement Learning - Context Detection (MARL-CD) allows to create more successful policies by analyzing the state and action traces of other agents and comparing their policies with the policies of rival agents; taking advantage of options that enable agents to select multi-actions within the same step; it differs in that it can learn some of the features (heuristics) in the environment states and make choices that will receive more rewards.

As a result of the experiments, it has been seen that agents learning with MARL-CD are more efficient in terms of detecting context change points more accurately and earlier, identifying contexts more accurately, consuming less energy and maximizing total average cumulative reward.

Explainability in machine learning [22] is a concept based on understanding machine learning models. The comprehensibility of the decisions made by models in real-life applications increases the reliability of some critical applications (e.g . healthcare, law). Models created with MARL-CD are interpretable. In other words, it can be explainable that why the agent chooses a policy or why the agent takes an action by looking at the partial models. But, in Deep Reinforcement Learning (DRL) uses deep neural networks. It has some shortcomings in terms of explainability [23]. MARL-CD is in a more advantageous position in terms of interpratibility.

During the experiments of this study, two rival agents were used. It has been observed theoretically and practically that MARL-CD works effectively for agents in two different teams. It has been observed that MARL-CD can tolerate up to 10% noise while monitoring other agents. In future studies, it is aimed to construct multi-agent environments competing in several teams. Thus, the development opportunity will be provided not only for two competing agents, but also for agents competing in teams. In addition, issues such as coordination of agents within the same team will also arise. This subject needs to be expanded by examining it with the involvement of game theory. Another aspect of MARL-CD that motivates us for further improvement is to

grow it more fault-tolerant to higher levels of noise that appears during inter-agent observation/communication.

The application of this study to real-life problems can be examined during future studies. Examples of these real-life environments are: AI economist environment [24] which includes agents and governments for learning the optimal economic policies; robot environment playing soccer in teams; on arcade games such as multi-car racing.

# 6 REFERENCES


[1] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning*. 1992, pp. 279–292.

[2] Andrew W. Moore and Christopher G. Atkeson. "Prioritized sweeping: Reinforcement learning with less data and less time". In: *Machine Learning* 13 (2004), pp. 103–130.

[3] Kumpati S. Narendra and Mandayam A. Thathachar. *Learning automata: An introduction*. Dover Publications, INC, 2012.

[4] Bruno da Silva et al. "Dealing with non-stationary environments using context detection". In: vol. 2006. Jan. 2006, pp. 217–224. DOI: 10.1145/1143844.1143872.

[5] Richard S. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1 (Aug. 1988), pp. 9–44. ISSN: 1573-0565. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009.

[6] Samuel Choi, Dit-Yan Yeung, and Nevin Zhang. "An Environment Model for Nonstationary Reinforcement Learning." In: Jan. 1999, pp. 987–993.

[7] Ezdin Aslanci et al. "Detection of regime switching points in non-stationary sequences using stochastic learning based weak estimation method". In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. 2017, pp. 787–792. DOI: 10.1109/INDIN.2017.8104873.

[8] B. Oommen and Luis Rueda. "Stochastic learning-based weak estimation of multinomial random variables and its applications to pattern recognition in non-stationary environments". In: *Pattern Recognition* 39 (Mar. 2006), pp. 328–341. DOI: 10.1016/j.patcog.2005.09.007.

[9] Hao Chen et al. "Detecting and Tracing Multi-Strategic Agents with Opponent Modelling and Bayesian Policy Reuse". In: *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)* (2020), pp. 1098–1103.

[10] Stewart Wilson. "Classifier Fitness Based on Accuracy". In: *Evolutionary Computation* 3 (May 1997). DOI: 10.1162/evco.1995.3.2.149.

[11] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. "Bayesian Policy Reuse". In: *Machine Learning* 104 (July 2016). DOI: 10.1007/s10994-016-5547-y.

[12] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (Dec. 2013).

[13] Longxin Lin. "Reinforcement learning for robots using neural networks". In: 1992.

[14] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533.

[15] D H Hubel and T N Wiesel. "Shape and arrangement of columns in cat's striate cortex". en. In: *J. Physiol.* 165.3 (Mar. 1963), pp. 559–568.

[16] Nikolaos Tziortziotis, Konstantinos Tziortziotis, and Konstantinos Blekas. "Play Ms. Pac-Man Using an Advanced Reinforcement Learning Agent". In: *Artificial Intelligence: Methods and Applications*. Ed. by Aristidis Likas, Konstantinos Blekas, and Dimitris Kalles. Cham: Springer International Publishing, 2014, pp. 71–83.

[17] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artif. Intell.* 112.1–2 (Aug. 1999), pp. 181–211. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(99)00052-1.

[18] Gail A. Carpenter, Stephen Grossberg, and David B. Rosen. "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition". In: *Neural Networks* 4.4 (1991), pp. 493–504. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90045-7. URL: https://www.sciencedirect.com/science/article/pii/0893608091900457.

[19] Gail A. Carpenter and Stephen Grossberg. "ART 2: self-organization of stable category recognition codes for analog input patterns". In: *Appl. Opt.* 26.23 (Dec. 1987), pp. 4919–4930. DOI: 10.1364/AO.26.004919. URL: http://opg.optica.org/ao/abstract.cfm?URI=ao-26-23-4919.

[20] *The Pac-Man Projects*. http://ai.berkeley.edu/project_overview.html. Accessed: 2022-05-28.

[21] Zijian Hu et al. "A Dynamic Adjusting Reward Function Method for Deep Reinforcement Learning with Adjustable Parameters". In: *Mathematical Problems in Engineering* 2019 (Nov. 2019), pp. 1–10. DOI: 10.1155/2019/7619483.

[22] Vaishak Belle and Ioannis Papantonis. "Principles and Practice of Explainable Machine Learning". In: *Frontiers in Big Data* 4 (July 2021), p. 688969. DOI: 10.3389/fdata.2021.688969.

[23] Yu Zhang et al. "A Survey on Neural Network Interpretability". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (2021), pp. 726–742. DOI: 10.1109/TETCI.2021.3100641.

[24] *The AI economist*. URL: https://www.salesforceairesearch.com/projects/the-ai-economist.

# CURRICULUM VITAE

## Education

| | | | |
|---|---|---|---|
| Msc | Marmara University | Computer Engineering | 2019 - 2022 |
| Bs | TOBB University of Economics and Technology | Computer Engineering(Double Major) | 2015 - 2018 |
| Bs | TOBB University of Economics and Technology | Electrical & Electronics Engineering | 2013 - 2018 |

## Work Experience

| | | |
|---|---|---|
| Trendyol | Software Engineer | Aug. 2020 - Present |
| Iyzico | Software Engineer | Aug. 2018 - Aug. 2020 |
| AYESAS | Software Engineer Intern | Sep. 2017 - Dec. 2017 |
| TAI | Software Engineer Intern | Jan. 2017 - Apr. 2017 |
| ICterra | Software Engineer Intern | May. 2016 - Aug. 2016 |