Titre: Title:	Non-Intrusive Load Monitoring: Event Detection Using Reinforcement Learning
Auteur: Author:	Mozaffar Etezadifar
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Etezadifar, M. (2023). Non-Intrusive Load Monitoring: Event Detection Using Reinforcement Learning [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/54397/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Non-Intrusive Load Monitoring: Event Detection using Reinforcement Learning

MOZAFFAR ETEZADIFAR
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Août 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Non-Intrusive Load Monitoring: Event Detection using Reinforcement Learning

présentée par **Mozaffar ETEZADIFAR**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Keyhan SHESHYEKANI, président
Jean MAHSEREDJIAN, membre et directeur de recherche
Houshang KARIMI, membre et codirecteur de recherche
Antoine LESAGE-LANDRY, membre
Hasan MEHRJERDI, membre externe

DEDICATION

To my beloved wife, who has unwaveringly stood by my side during the most challenging of days, and to my cherished family, whose support has been a constant source of strength, I dedicate this thesis.

ACKNOWLEDGEMENTS

I would like to express my deep sense of gratitude to my supervisors Dr. Jean Mahseredjian and Dr. Houshang Karimi for their supervision and support of this Ph.D. project. I would also like to express my warm gratitude to all those at Polytechnique Montreal, the jury members, and the president who have made this thesis possible: Dr. Hasan Mehrjerdi, Dr. Antoine Lesage-Landry, and Dr. Keyhan Sheshyekani.

Mozaffar Etezadifar, August 2023.

RÉSUMÉ

La surveillance de charge non-intrusive (SCNI) est la technique utilisée pour dissocier la consommation d'énergie totale en utilisation d'énergie individuel au niveau des appareils sans avoir besoin de capteurs ou de compteurs supplémentaires. SCNI est essentielle pour la mise en œuvre de certaines techniques de gestion de la demande (GD) dans les réseaux électriques. Les algorithmes SCNI comprennent plusieurs modules et étapes, parmi lesquels l'étape de détection d'événements ayant un impact significatif sur les résultats finaux. Cette thèse propose un nouvel algorithme et une architecture pour la détection d'événements dans les solutions SCNI basées sur des événements afin d'améliorer leurs performances.

Les algorithmes de détection d'événements existants, tels que les méthodes probabilistes ou heuristiques expertes, reposent sur des critères d'entrée spécifiques pour identifier les transitions de charge particulières pour lesquelles ils ont été conçus. Cependant, les algorithmes traditionnels de détection d'événements échouent souvent à détecter plusieurs événements de charge dans les profils de charge résidentiels. Pour réduire la dépendance à des conditions spécifiques et améliorer les performances des algorithmes de détection d'événements, nous utilisons l'apprentissage par renforcement (AR) pour former un agent qui apprend et intègre les connaissances des autres algorithmes de détection d'événements avec lesquels il est formé. Grâce à de nombreuses interactions entre l'agent AR et les algorithmes de détection d'événements intégrés dans son système de rétroaction, l'agent AR acquiert les connaissances de toutes les méthodes traditionnelles et améliore leurs performances tout en minimisant les erreurs.

En ce qui concerne l'ensemble de données, nous avons choisi l'ensemble de données iAWE du monde réel parmi les ensembles de données SCNI disponibles publiquement. Cet ensemble de données a été collecté dans une maison à New Delhi, en Inde, et comprend des données sous-comptées pour plusieurs appareils, ainsi que la consommation d'énergie totale échantillonnée à une fréquence de 1 Hz. Cependant, comme il ne comprend pas d'étiquettes d'événements de charge, un algorithme générateur de profil de charge artificiel est conçu pour générer des profils de charge étiquetés illimités à partir des signatures de charge iAWE pour le processus d'apprentissage par renforcement et l'évaluation.

Pour améliorer davantage les performances de l'agent AR, nous introduisons une nouvelle structure de mémoire appelée Double Mémoire de Relecture (DMR), qui joue un rôle crucial dans la capacité de l'agent AR à détecter des événements. Sans la structure DMR, l'agent AR rencontre des problèmes lorsqu'il est confronté à un nombre illimité d'états possibles dans

un profil de charge.

Pour étudier les performances de la méthode proposée de détection d'événements pour SCNI en utilisant l'agent AR (ARSCNI), nous avons défini quatre groupes de scénarios impliquant différentes combinaisons de signaux d'entrée, de fréquences d'échantillonnage et d'informations externes. L'ensemble de signaux d'entrée comprend toutes les combinaisons de puissance active (P), de puissance réactive (Q) et de courant (I). L'ensemble de fréquences d'échantillonnage contient des fréquences de 1 Hz et 0,2 Hz. L'ensemble d'informations externes indique la présence ou l'absence de scores de performance des algorithmes traditionnels de détection d'événements. La méthode de détection d'événements ARSCNI proposée a surpassé tous les algorithmes intégrés de détection d'événements traditionnels dans tous les différents groupes de scénarios.

Étant donné que l'architecture proposée n'impose aucune exigence quant au fonctionnement des algorithmes de détection d'événements existants et améliore leur flexibilité en termes de critères opérationnels, la méthode de détection d'événements ARSCNI peut éliminer de nombreux obstacles à l'utilisation généralisée de ARSCNI dans les réseaux électriques.

ABSTRACT

Non-intrusive load monitoring (NILM) is the technique used to disaggregate total energy consumption into individual appliance-level energy usage without the need for additional sensors or meters. NILM is vital for implementing certain demand-side management (DSM) techniques in power grids. NILM algorithms consist of several modules and steps, among which the event detection step significantly impacts the final results. This thesis proposes a novel algorithm and architecture for event detection in event-based NILM solutions to enhance their performance.

Existing event detection algorithms, such as probabilistic or expert heuristic methods, rely on specific input criteria to identify particular load transitions for which they have been designed. However, traditional event detection algorithms often fail to detect several load events in residential load profiles. To reduce dependence on specific conditions and improve the performance of event detection algorithms, we employ reinforcement learning (RL) to train an agent that learns and integrates the knowledge of other event detection algorithms it is trained with. Through numerous interactions between the RL agent and embedded event detection algorithms in its feedback system, the RL agent learns the knowledge of all traditional methods and enhances their performance while minimizing errors.

Regarding the dataset, we chose the real-world iAWE dataset from publicly-available NILM datasets. This dataset was collected from a house in New Delhi, India, and includes sub-metered data for several appliances, as well as total power consumption sampled at a frequency of 1 Hz. However, as it does not include load event labels, an artificial load profile generator algorithm is designed to generate unlimited labeled load profiles from iAWE load signatures for the reinforcement learning training process and evaluation.

To further enhance the RL agent's performance, we introduce a new memory structure called Dual Replay Memory (DRM), which plays a crucial role in the RL agent's ability to detect events. Without the DRM structure, the RL agent encounters problems when facing an unlimited number of possible states in a load profile.

To investigate the performance of the proposed event detection method for NILM using the RL agent (RLNILM), we defined four groups of scenarios involving different combinations of input signals, sampling frequencies, and external information. The input signals set includes all combinations of active power (P), reactive power (Q), and current (I). The sampling frequency set contains frequencies of 1 Hz and 0.2 Hz. The external information set indicates the presence or absence of performance scores from traditional event detection algorithms.

The proposed RLNILM event detection method outperformed all embedded traditional event detection algorithms in all different scenario groups.

Since the proposed architecture does not impose any requirements on the operation of existing event detection algorithms and enhances their flexibility in terms of operational criteria, the RLNILM event detection method can eliminate many obstacles to the widespread use of NILM in power grids.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

NILM	Non-intrusive Load Monitoring
DSM	Demand Side Management
ML	Machine Learning
RL	Reinforcement Learning
ED	Event Detection
CL	Confidence Level
LLR	Log Likelihood Ratio
SWDC	Sliding Window Distribution Change
P	Active Power
Q	Reactive Power
I	Current
s	State
a	Action
r	Reward
ω	Window length
θ_P	Power threshold
θ_V	Voting threshold
θ_δ	Standard deviation threshold
θ_u	Uniqueness threshold
TP	True Positive
FP	False Positive
FN	False Negative
γ	Discount factor
α	Mix rate
π	Policy
ϵ	Exploration rate
$Q_{(s,a)}$	State action value
V_s	State value

CHAPTER 1 INTRODUCTION

In this chapter, we briefly explain what NILM is and why it matters. Then, the motivation for improving event detection algorithms in NILM solutions and the general approach is explained. A short literature review on event detection algorithms is presented in the next sections. Following that, the research contributions and the dissertation outline are stated.

1.1 Context

Over the past decades, the world's population has quadrupled while energy consumption has increased 16 times [1]. This ever-increasing energy consumption is accompanied by many environmental effects including high carbon emission which leads to devastating natural phenomena, e.g., global warming. Moreover, fulfilling this energy demand, in particular, the demand for electrical energy, has become a challenge for many governments all around the world [2]. Constructing more power plants may temporarily alleviate the energy shortage problem, however, it is neither a sustainable nor environment-friendly solution. In the long term, it will exacerbate the problem.

The advent of new electrical appliances with higher power consumption, the augmentation of the urban population, and the modern lifestyle have made the importance of the residential sector bolder than ever for the energy industry [3]. In the United States, the residential sector was responsible for 34% of the power consumption by 2010 which will reach 39% by 2030 [4]. In Canada, residential units consume around 32% of the electrical energy [5]. In European Union (EU), households consume 30% of the entire energy generation which emits 36% of total CO_2 in EU [6].

Aside from more generation which poses its own problems, Demand Side Management (DSM) is a solution to satisfy the increasing demand for electricity, by the existing generation capacity [7]. An effective DSM improves the grid's overall efficiency [8] while mitigating energy shortage problem [9]. DSM consists of several techniques some of which involve sending feedback to energy consumers. According to a study [10] indirect feedback, which occurs after consumption, such as household enhanced billing or daily/weekly feedback leads to 3.8% and 8.4% percent of annual energy saving respectively. The same study suggests that consumption direct feedback which happens in real-time can increase this saving by up to 12%. In [11] the author points out that real-time personalized feedback down to the appliance level in households can result in more than 12% annual energy saving. Appliance level feedback

depends on accurate Load Monitoring (LM).

LM is the key to achieve an effective and precise DSM [12]. By LM, one can realize how much energy each load type in any target area is consuming and how it can be optimized by providing real-time feedback. Moreover, consumers' electricity consumption behavior can be better understood which facilitates load forecasting, planning, and policy-making for the grid [3] [13] [14]. On the other hand, on the consumption side, this optimization is reflected in reduced electricity bills with almost the same comfort level [15]. LM is categorized into Intrusive Load Monitoring (ILM) and Non-intrusive Load Monitoring (NILM). In ILM, sensors are installed on target appliances to monitor their consumption. This approach results in a very accurate consumption analysis. However, it has some drawbacks that prevent it from being extensively used in the grid [6]. ILM implies a sense of privacy infringement and house owners may feel that their daily routines and activities are being monitored. The cost of installing many measurement devices on different appliances can be significant. In addition, not all power consumer appliances can be tracked by installing a meter on them, e.g., ceiling lamps or plug-in devices. On the other hand, in the NILM method, a single-point measurement is used to disaggregate each appliance's power consumption. NILM removes many of the ILM's obstacles to the practical use of LM in the power grid. Many studies have been conducted to improve NILM performance since its initial development by George Hart in 1992 [16].

There are many viewpoints from which NILM algorithms can be studied. Most NILM algorithms include principal steps, e.g., data acquisition, event detection, feature extraction, load classification, energy decomposition, and performance evaluation [6]. In data acquisition, the first step is to choose a dataset. It can be one of the available online public datasets, a lab-made private dataset, or an artificially-made dataset (using real appliances' load signatures). Each dataset contains specific electrical signals. The data sampling frequency plays a key role in this step. By low sampling rates, in most cases, signals, e.g., active power (P), reactive power (Q), voltage (V), and current (I) are being used [17] [18] [19]. In contrast, with high data sampling frequencies transient state analysis becomes an available option, e.g., studying harmonics [20], voltage noise on the power line [21], or frequency responses [22]. Low-frequency methods have attracted more attention because most smart meters installed around the world sample data at rates lower than 1 sample per second. On the other hand, the higher the frequency the more expensive the meter becomes [23] [24]. The next important step in NILM algorithms is event detection. Event detection is the cornerstone of many NILM algorithms. All next steps are built on top of the event detection step. Thus it is worth focusing on this step and trying to improve it. There are currently different types of event detection algorithms, e.g., Expert Heuristic [25], Log Likelihood Ratio [26], and Matched

Filter [27]. Each method has its own cons and pros, thus, there is no one-fit-all solution for all the conditions. Each method should be properly tuned for the target load profile to accurately identify load transitions (turn ON/OFF moments) in a load profile while avoiding mistakenly detecting fluctuations as an event. The next step is feature extraction in which we try to calculate different statistical variables of the detected load transition in order to cluster or classify transition moments. In the next step, we use these features to classify load events and decompose the energy consumption. The variety of algorithms in this step is significant. Probabilistic procedures like Hidden Markov Model (HMM) [28] became popular because of their good performance, however, they are computation-intensive and have some limits for real-time analysis. Supervised classification algorithms need a tagged library of appliance load signatures which makes their application limited despite their good performance [22]. Unsupervised clustering algorithms have gained popularity due to their ease of use, however, there is still the need for user interaction for final appliance classification [13]. With the advent of high-performing processors, Artificial Neural Networks (ANNs) became one of the best-performing energy decomposition algorithms, e.g., seq2point, seq2seq, etc., however, they need a significant amount of training. Zeifman *et al.* [29] and Zoha *et al.* [30] have published detailed review articles on these algorithms. Lastly, for the performance evaluation of different NILM algorithms, different studies suggest various metrics. The most important metric for the entire NILM process is the error of energy disaggregation for each appliance, however, for the event detection step, it could be $F_1 score$ which shows how accurate the event detection algorithm is.

It was a brief introduction to NILM solutions and their important role in energy optimization in the grid. In the next section, the main motivation behind our proposed solution is discussed.

1.2 Motivation

As it is mentioned earlier, the event detection algorithm plays a key role in the process of a NILM solution. Having an accurate and fast event detection algorithm will improve the entire NILM process significantly. However, every NILM algorithm has its own parameters and limits. Every event detection algorithm should be fine-tuned to have optimal performance on each load profile. Some of them are only functional if some minimum requirements are met, e.g., a high-frequency data sampling or need for a specific electrical signal as their input. Considering that each algorithm is able to work well for detecting its own targeted appliances when its required conditions are met, we propose an algorithm that learns the knowledge of each of these event detection algorithms and combines their knowledge of identifying different

appliances' events in a single model with improved performance. This approach helps us focus more on designing event detection algorithms that can identify complex load events (load signatures) without sacrificing the ability to identify simpler load events. In fact, our proposed method is able to combine several simpler algorithms and make a versatile model that performs better event detection than those algorithms with which it has been trained. This goal is achieved by using Reinforcement Learning (RL) which is a subset of Machine Learning (ML). In the bigger picture, beyond the event detection step of a NILM algorithm, we can use RL to improve the entire NILM process. A well-designed RL structure reads the grid signals as the input states and trains an RL agent which is able to determine how much energy each appliance at that moment is consuming. The amount of energy consumed by each load is what the user expects from a NILM algorithm. If we have the outputs of other traditional NILM algorithms for the same input states at the same time, the proposed RL agent can compare its performance with those traditional algorithms. Based on this comparison we can calculate the energy estimation error for all the loads, and use that as an input signal for the agent. This feedback relates the next state to the previous action of the RL agent which makes this case a complete RL algorithm problem. Despite this possible design, we decided to only focus on the event detection step to investigate the feasibility of performance improvement on such a problem. The full implementation of the proposed algorithm on the NILM process remains a great future work option.

1.3 Literature review

In Section 1.1 the entire NILM topic was briefly reviewed. As the focus of this research is improving the event detection algorithms in the NILM solutions, in this section we go through different event detection algorithms in the NILM literature to see how different event detection algorithms perform.

1.3.1 Event-based vs. Non-event-based NILM

The input of NILM can be various electrical signals, e.g., active power (P), reactive power (Q), voltage (V), or current (I), voltage-noise, V-I trajectory, harmonics, etc. Low sampling frequency will limit the usage of some of these electrical signals such as harmonics as it needs high-resolution data to be used. Some NILM algorithms use these inputs when the electrical signal is in a steady state while some other algorithms analyze the transient state of these signals. The transient state happens to be between two steady states. A transient state usually is a sign of load activation (turn ON/OFF moment) or an internal change of state in an appliance. Thus, from now on these transitions are called the events. The algorithms

that analyze the steady states of a load profile (non-event-based) are computation-intensive as they should process the input data all the time. In contrast, event-based algorithms, are activated only when they detect an event in a load profile. Therefore, event-based algorithms are more practical solutions to be used at scale. The next section reviews some of the most important event detection algorithms.

1.3.2 Event detection algorithms

In the literature, there are many innovative methods to identify events of a load profile. It means that the input of the event detection algorithm is one or more electrical signals sampled at a certain frequency, and the output of the event detection algorithm is the vector e which includes the timestamp of every detected event.

$$\mathbf{e} = \{e_{t1}, e_{t2}, ..., e_{tn}\} \tag{1.1}$$

Some event detection algorithms need to be trained to detect targeted events, however, there exist many methods that can identify load events with no need for prior information. Event detection algorithms can be grouped into three main categories [26]: i) expert heuristic, ii) probabilistic methods, and iii) matched filters.

1.3.2.1 Expert heuristics

This category of event detection algorithms is probably the most basic one. In these algorithms, the time series of input data is scanned, and the algorithm searches for changes above a certain threshold. George Hart introduced such an event detection function in his work in 1985 which analyzed both active and reactive power to identify events [16]. In most event detection algorithms the input data is filtered to ignore the noise in the input data and avoid creating false positives. Then, the load events are identified by comparing the absolute difference between two adjacent data points and choosing the time stamp in which the difference is above a certain pre-defined threshold [31]. In more complex algorithms, some adjustments were done to improve the algorithm's performance. For example, instead of comparing the difference between two consecutive values, two samples with n data points in between are compared to avoid transient fluctuations [32]. Another adjustment is to consider a window around every detected event in which no other event can be flagged, thus no false positive is made close to another detected event. In some heuristic algorithms, P and Q are used together to increase the certainty of the event detection algorithm's output.

1.3.2.2 Probabilistic methods

One of the best-performing event detection algorithms is probabilistic methods. In this category, the event detection process happens in two steps. First, the probability of being an event is calculated for every time step within a sliding window over the input signal (in most cases it is P). This probability is referred to as the detection statistics. These detection statistics can be computed by many statistical methods, e.g., Goodness-of-fit (GOF) [33], Cumulative Sum (CUSUM) [34], Generalized Likelihood Ratio (GLR) [35], or other mathematical methods, e.g., Kernel Fisher Discriminant Analysis (KFDA) [36]. In the next step, these calculated statistics for every data point, are compared against a threshold. Any timestamp with detection statistics higher than the threshold can be considered as an event [33,35]. However, there are more robust methods for this second step, e.g., voting algorithm or finding the local maxima between detection statistics which makes the probabilistic methods, robust and reliable event detection algorithms [37].

1.3.2.3 Matched filters

In this category, the event detection algorithm identifies the events by correlating a known signal to an unknown one to detect if the known signal is present in the load profile or not. In other words, these algorithms are searching for the known appliance transients in the aggregate load profile using various filters. For example, in [38, 39], two filters were used. The first filter finds the shape of the known transients in the aggregate load profile and the second filter makes sure that the transient is not mistaken by some random noise. In fact, matched filters emphasize the potential load events while they depreciate the steady states. For example, in [40] the authors have used Hilbert transform to the current sampled at the frequency of 20 kHz. Combined with average and derivation filters, the authors managed to extract the load events from the aggregate load profile.

As the matched filter algorithms need some prior knowledge about the transient states of target appliances, one can make an analogy between supervised Artificial Neural Networks (ANN) methods and matched filters. Both these methods need to have a database of transient states of the target appliance to be able to identify them in the aggregate load profile. However, the way they fulfill this task is different. The supervised ANN method can perform well on the datasets with which it has been trained, but as it is highly dependent on its training dataset, we do not mention Supervised Neural Networks as an independent event detection category.

All of these algorithms need to be trained or their parameters should be fine-tuned based on

the total load profile that they are analyzing. The goal is to identify the most true positive events possible while avoiding false positives. However, there is always kind of a sacrifice while optimizing the algorithms. An optimized algorithm for a specific load profile may not be able to operate acceptably on another load profile. Nonetheless, we should try to design an event detection algorithm that is able to identify the most possible events considering all the variables of the problem.

1.4 Research contribution

This research is aimed to propose a new architecture that improves the performance of event detection algorithms for the NILM purpose. Using the proposed method, different event detection algorithms can cooperate in a way that leads to the training of a versatile event detection algorithm. In fact, this algorithm learns the knowledge of other event detection methods and combines them in a single model that can outperform those mentioned algorithms. Thus, the main contributions of this research can be listed as follows:

- Proposing a new architecture based on the RL that can learn the knowledge of existing event detection algorithms and outperform them,
- proposing the Dual Replay Memory (DRM) for the RL structure which makes it possible for the RL algorithm to overcome complex situations, e.g., NILM problems,
- Utilizing RL for the first time in a load monitoring problem that makes it more flexible and extensive in real-world cases.

1.5 Thesis structure

The rest of this thesis is structured as follows. In Chapter 2, the chosen dataset for this research is introduced and its attributes are discussed. Chapter 3, covers the event detection techniques that are used in this work. In Chapter 4, the fundamental machine learning concepts that are used in some NILM algorithms are briefly explained. Chapter 5, thoroughly explains the proposed architecture and algorithm. In Chapter 6, the proposed method is tested under different conditions and scenarios and the results are discussed. Chapter 7 concludes the thesis and presents a summary of the chapters. The limits and future works are also discussed in Chapter 7.

CHAPTER 2 DATA

Data is the cornerstone of every ML algorithm. Without an appropriate dataset, it is not possible to train, test, or evaluate the performance of our algorithms. The importance of having a reliable dataset becomes bolder when it comes to NILM algorithms as they are supposed to disaggregate different datasets (load profiles). Because our topic is NILM, the focus of this thesis is on those aspects of datasets that are used in such studies.

2.1 Datasets

Datasets that are used for the NILM purpose, can be categorized from different viewpoints. In the simplest case, a dataset for the NILM purpose consists of an aggregate electrical signal of a phase in the target room (or residential unit). However, datasets usually contain more data than that. These aspects can be itemized as follows:

- **Electrical signal**: Different signals can be measured, e.g., active power (P), reactive power (Q), apparent power (S), current (I), voltage (V), phase angle (θ), or frequency (f). Moreover, some datasets provide additional non-electrical data, e.g., occupancy, ambient condition, water consumption, temperature, or fuel consumption.

- **Sampling frequency**: Sampling frequency is of the key attributes of every dataset. In NILM literature the frequencies above 1 Hz are considered high and below that are referred to as low, however, there is no specific rule for that. Higher frequency provides more information and makes it possible to implement some event detection algorithms based on harmonics or other transient state features. Nonetheless, high-frequency sampling comes with its cost including the need for higher bandwidths to transfer the data, the need for higher storage systems, and more expensive sampling devices. Low-frequency sampling is a better option for data transfer and storage, however, it hinders us from implementing some event detection algorithms. It should be noted that most existing smart meters in power grids around the world are of this class. Thus, one should consider the pros and cons of every frequency range for their specific problem.

- **Length of measurement**: Having an adequate number of data samples is necessary for training and test steps in many NILM algorithms. The period of data sampling should be long enough in order to have enough data samples for training the algorithm

(e.g., supervised algorithms) and meaningful evaluation of the results. There exist public datasets in which the electrical signals are sampled from a few hours to more than two years.

- **Variety of appliances**: Another important aspect of datasets for the NILM purpose is the variety and number of appliances in them. A house with many electric appliances creates a more complex load profile, however, it is more informative. On the other hand, sampling in different houses will provide the researchers with a variety of load signatures of the same appliances due to their different models. Having more models of the same appliance in the datasets is beneficial to train more expandable NILM solutions.

- **Labeling**: The presence of labeling in the dataset is also of great importance. For the purpose of labeling, the dataset creators use a smart plug for each of the targeted appliances. The smart plugs record the electrical signals of their connected appliance. Thus, in labeled datasets, either every targeted appliance has its own load profile from which we can see the activity period and power consumption values, or the timestamps of turning ON/OFF moments are recorded. The labels are crucial for training or evaluating purposes in NILM solutions.

Considering the approach for creating the datasets, they can be categorized into three different groups: i) public datasets, ii) private (lab-made) datasets, and iii) simulated (artificially-made) datasets. There are several public datasets that are available online for free, however, their attributes may not match the requirements of different NILM studies. Therefore, some researchers create their own datasets in the lab using different appliances and sensors. Another method that some researchers choose to have the desired dataset for their NILM study is to simulate the load profile using real-world load signatures from the public NILM datasets. The superiority of this method over other methods is that it is possible to create as many data samples as we need. Thus, there is no limit on the available samples for the training and evaluation steps. Several studies have investigated available public datasets and their attributes. In this experiment, we used the iAWE dataset and created a simulated dataset based on iAWE real-world data.

2.2 iAWE dataset

In this research, iAWE dataset is chosen as the main source of data. iAWE dataset presented in [41] contains the data samples of energy and water consumption of a house in New Delhi, India over 73 days. The authors of [41] have used different sensors to measure the electrical

signals of various appliances in the target residential unit. Electrical signals, e.g., active power P, reactive power Q, apparent power S, voltage V, current I, and power factor were sampled at the rate of 1 Hz frequency. Even though each appliance is monitored separately, there is no labeling for load transitions (turn ON/OFF moments).

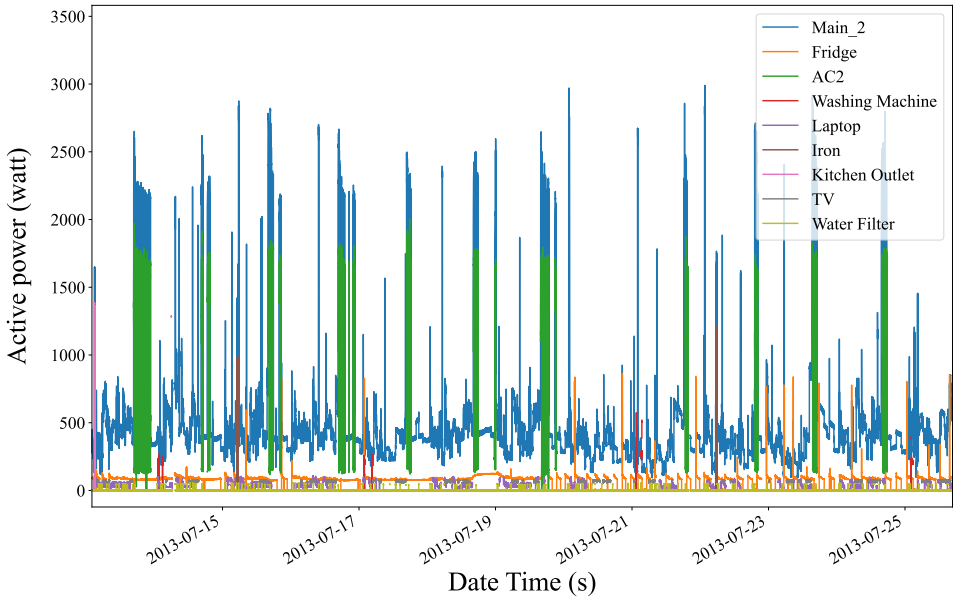

Figure 2.1 Aggregate load profile of iAWE dataset

Despite 73 days of data, there are many periods of missing data due to the sensors' problems or connectivity issues. Thus, to have enough density of different load signatures of the appliance, we tried to simulate the load profile using the real load signatures of the appliance in the iAWE dataset. Artificial load profiles make it possible to create precise labeling for load transition moments which is necessary for the method of learning that is proposed in the next chapters.

2.3 Artificial data generation

In the iAWE dataset, nine different appliances are monitored, including a fridge, an air conditioner, a washing machine, a laptop, an iron, a kitchen outlet, a television, a water filter, and a water pump. The appliances are chosen if they have enough repetition during the period of measurement so their load signature samples have enough variety for load profile reconstruction purposes. Using an algorithm all the load signatures of every appliance are extracted, i.e., the data samples of each appliance have been extracted from the turn-ON moment to the turn-Off moment. Each load signature was then saved as a Comma Separated

Values (CSV) file. After that, a load profile reconstructor algorithm was used to merge all the load signatures of different appliances with the distribution of a normal house to form the final aggregate load profile. This method of generating load profiles enabled us to create as many unique load profiles as needed while having precise labeling for load transitions of each appliance. It is noteworthy that a base load and Gaussian noise with the same distribution as in the iAWE dataset are added to the final outputs to make them as realistic as possible.

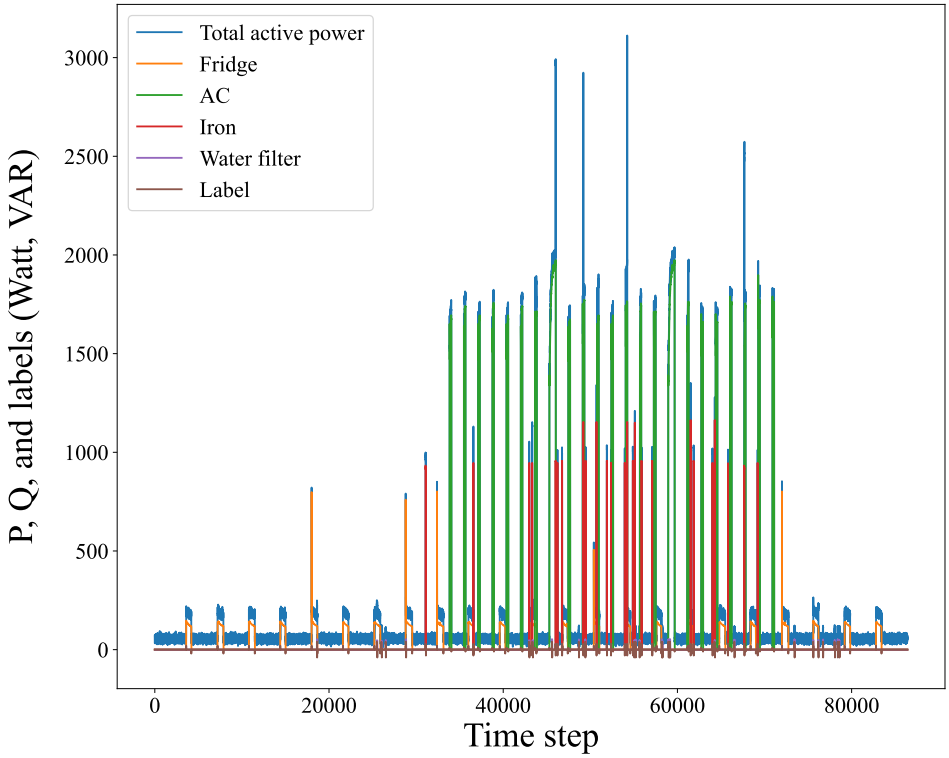

Figure 2.2 Artificially-made load profile using iAWE dataset

2.4 Labeling

For training and evaluation purposes it is necessary to have informative labeling. Using our load profile generator, we labeled all the turn-ON and turn-OFF moments of all load signatures with a numeric value. Therefore, for every time step of the generated load profile, the type of the appliance and the type of load transition (turning ON or OFF) is included in that value. This vector of values is called the key vector. The following table represents the coding used in the key vector.

Table 2.1 Key vector coding for iAWE load signatures

Load event	
AC ON	0
Fridge ON	1
Water filter ON	2
Iron ON	3
No event	4
AC OFF	5
Fridge OFF	6
Water filter OFF	7
Iron OFF	8

CHAPTER 3 EVENT DETECTION

In this chapter, we introduce two event detection algorithms that we have used in our experiment. Every event detection algorithm is supposed to accept some inputs, most commonly, electrical signals. Then the data is pre-processed to be prepared for event detection. The algorithm identifies the timestamps in which a probable load transition is present and outputs those timestamps as a vector to which we refer as the events vector. In this research, we have used two of the most popular event detection algorithms which are widely used in the NILM literature [42]. In the following sections, we introduce Log-Likelihood Ratio detector (LLR) and Expert Heuristic (EH) event detection algorithms.

3.1 Log likelihood ratio detector

In event-based NILM algorithms, the event detection algorithm is usually designed based on the fact that a load transition is associated with an abrupt change in the values of electrical signals. For the sake of simplicity, it is assumed that the input signal is active power P. The event detection algorithm takes P values as the input, and outputs events vector, $\mathbf{e} = [e_1, ..., e_M]$, that contains the timestamps of the identified load events. Figure 3.1 illustrates some load events in our dataset. The change in the level of P represents a load event. If the event detection algorithm is able to successfully identify the presence of a load event at a specific distance from the actual timestamp (or index) of the event, then we can consider the event detection successful. LLR is the general name of a group of event detection algorithms; LLR, LLR voting, and LLR maxima to name a few. They work by comparing the distribution of data points around the target data sample. In this section, we explain LLR voting and its working principles.

It is assumed all the data points of the load profile are already stored and we have access to all of them. Thus, it is an offline or delayed version of the LLR event detection algorithm, however, with some modifications we can have an online version. As we mentioned earlier, Figure 3.1 illustrates examples of load events in our dataset. Active power values at different indices are indicated as $P[n]$ where $n = 1, ..., N$ and the sampling frequency is f_s. The data point that is under analysis is referred to as the test point. The goal is to determine if the test point ($P[n]$) is more like w_1 points after it, or if it belongs to w_0 data points before. This likelihood ratio can be determined by calculating the Gaussian distribution of these pre-windows and post-windows (mean and standard deviation). Thus, these attributes of every data point is called test statistics ($S[n]$, $n = w_0 + 1, ..., N - w_1$) and they can be formulated

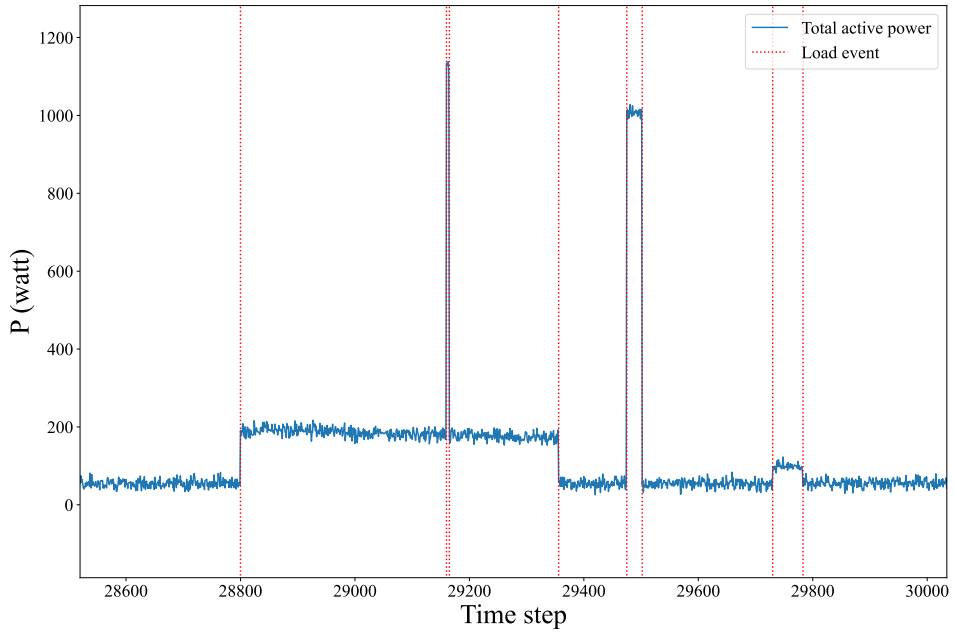

Figure 3.1 Examples of the load events in the dataset

as the log of the ratio of the likelihood that test point ($P[n]$) belongs to post-window over that of the pre-window:

$$S[n] = ln\left(\frac{N(P[n]|\mu_{1,n}, \sigma_{1,n})}{N(P[n]|\mu_{0,n}, \sigma_{0,n})}\right) = ln\left(\frac{\sigma_{0,n}}{\sigma_{1,n}}\right) + \frac{(P[n] - \mu_{0,n})^2}{2.\sigma_{0,n}^2} - \frac{(P[n] - \mu_{1,n})^2}{2.\sigma_{1,n}^2} \tag{3.1}$$

where N indicates the normal distribution and $\mu_{1,n}$, $\sigma_{1,n}$, $\mu_{0,n}$, and $\sigma_{0,n}$ represent the mean and standard deviation of data points in pre-windows and post-windows, respectively, i.e. :

$$\mu_{0,n} = \frac{1}{w_0} \sum_{k=n-w_0}^{n-1} P[k] \tag{3.2}$$

$$\mu_{1,n} = \frac{1}{w_1} \sum_{k=n+1}^{n+w_1} P[k] \tag{3.3}$$

$$\sigma_{0,n}^2 = \frac{1}{w_0 - 1} \sum_{k=n-w_0}^{k-1} (P[k] - \mu_{0,n})^2 \tag{3.4}$$

$$\sigma_{1,n}^2 = \frac{1}{w_1 - 1} \sum_{k=n+1}^{n+w_1} (P[k] - \mu_{1,n})^2 \tag{3.5}$$

Considering the length of pre-widnow and post-window (w_0 and w_1 respectively), we note that for $n \leq w_0$ and $n \geq N - w_1$ the values of mean μ and σ can not be calculated so to make the length of P and S equal, $S[n]$ is defined as follows:

$$S[n] = \left\{ \begin{array}{lll} 0 & , & n < w_0 + 1 \\ ln\left(\frac{\sigma_{0,n}}{\sigma_{1,n}}\right) + \frac{(P[n]-\mu_{0,n})^2}{2.\sigma_{0,n}^2} - \frac{(P[n]-\mu_{1,n})^2}{2.\sigma_{1,n}^2} & , & w_0 + 1 \leq n \leq N - w_1 \\ 0 & , & n > N - w_1 \end{array} \right\} \tag{3.6}$$

Having test statistics, we create another vector called modified log-likelihood ($l[n]$). To create $l[n]$, every $S[n]$ value whose difference between pre-windows and post-windows means is lower than a certain threshold, θ_P, is forced to be zero. Thus, vector $l[n]$ is defined as follows:

$$l[n] = \left\{ \begin{array}{ll} S[n] & |\mu_{1,n} - \mu_{0,n}| \geq \theta_P \\ 0 & \text{else} \end{array} \right\} \tag{3.7}$$

The likelihood value, $l[n]$, shows how significant the change of the P values distribution is. For example, if a steady state period of the load profile is considered, the distribution of P in both pre-windows and post-windows are almost the same (a steady value plus a Gaussian noise), thus the log-likelihood of that data point will be close to zero. In contrast, if the data point of interest is an event point, the difference in the distribution of pre-windows and post-window values will have a higher than one absolute, therefore its log-likelihood becomes a noticeable value in $l[n]$ vector. In the simplest LLR approach, any likelihood ratio, higher than a certain threshold can be identified as an event. However, there exist more accurate methods to identify probable events, e.g., LLR voting compute-intensive.

In the LLR voting scheme, a voting window with the length w_V slides over the log-likelihood vector $l[n]$. It is important to mention that only the absolute values of $l[n]$ are considered and not the signs. With every shift, a vote is assigned to the index n in which the $l[n]$ has the highest value across the voting window. If all the values are equal, no vote is assigned. This is called the votes vector, V. Thus, the n^{th} value of V indicates how many votes the data point with index n has gained. So far, three different windows, pre-window (w_0), post-window (w_1), and the voting window (w_V) have been defined. Therefore, some data points at the beginning and end of the data stream are actually not processed due to the length of the windows. As the NILM datasets contain numerous data points, these few samples can be

ignored safely. Figure 3.2 illustrates how all the mentioned vectors are formed side by side.

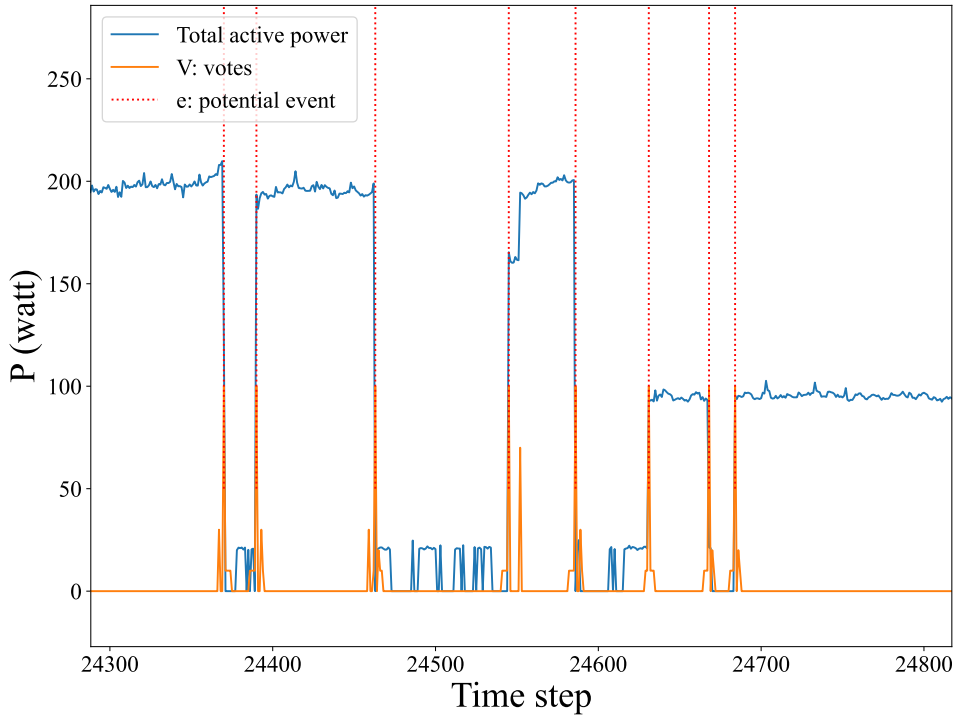

Figure 3.2 LLR voting vectors

The votes vector shows us how probable it is for every index to be the index of an event. The voting threshold is defined as θ_V. Any index in V with an equal or higher number of votes than θ_V can be considered an event. It should be noted that θ_V is a number less than or equal to w_V because the maximum number of votes that an index can gain is w_V (one vote in each shift of the voting window). Thus, the final vector of events is defined as follows:

$$\mathbf{e} = \{n \in \{1, ..., N\} : V[n] \geq \theta_V\} \tag{3.8}$$

It wraps up the concept of the LLR voting event detection algorithm. In the next section, another popular event detection algorithm in NILM literature is discussed.

3.2 Sliding Window Distribution Change

In expert heuristic event detection methods, the sliding window technique is commonly used. The sliding window scans the total load data and performs calculations on those data points to find out if there an event happened in that window or not. Usually, a significant change in the distribution of the data points in the sliding window indicates an event. In this section,

we introduce a computationally-efficient event detection method using active power (P) as its input stream.

In this method, after preprocessing the aggregated load data to eliminate rush power spikes and fluctuations, the difference between consecutive data samples is calculated as below:

$$\Delta P_t = P_{t+1} - P_t \tag{3.9}$$

This differential load profile is much easier for the event detection algorithm to handle than the original data points as the steady states become almost zeros and power changes become more visible to the algorithm. Figure 3.3 illustrates preprocessed and differential power profiles.

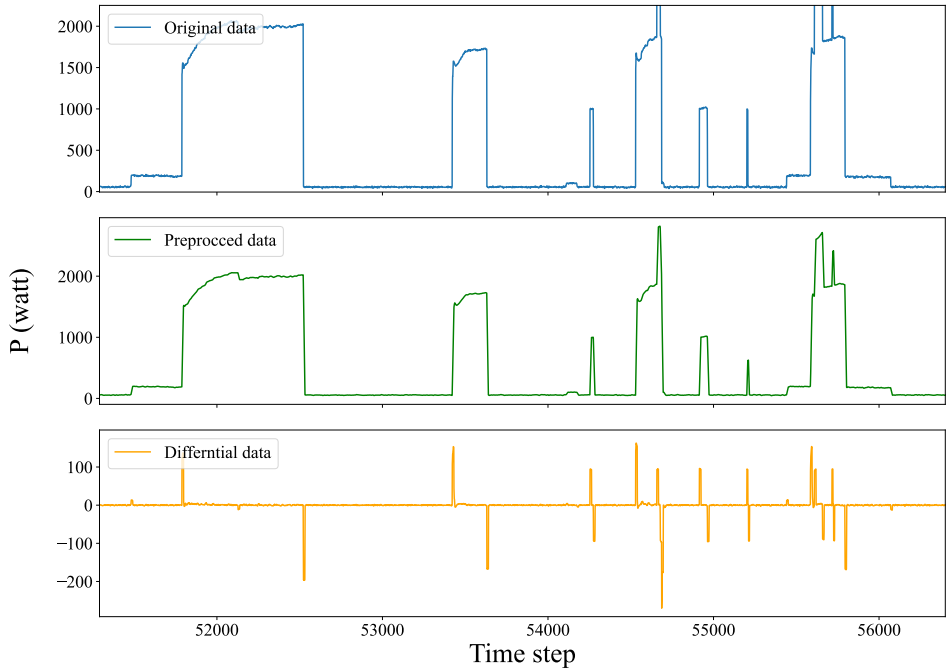

Figure 3.3 Raw data versus differential data

In this step, the size of the sliding window, w_s, should be defined. The length of the sliding window determines the sensitivity of the event detection algorithm. Increasing w_s lowers the sensitivity of the algorithm while reducing its speed. On the other hand, a narrow sliding window becomes more vulnerable to false positives. Therefore, like any other event detection algorithm, an optimization process based on the target load profile is necessary for the proper performance of the algorithm. Having w_s, the standard deviation of the data points in every window should be calculated as below:

$$\sigma_{n,n+w_s} = \sqrt{\frac{1}{w_s}\sum_{k=n}^{n+w_s}(\Delta P[n] - \mu_{n,n+w_s})^2} \tag{3.10}$$

where n is the index of the data point at the beginning of the sliding window and μ is the average of data points inside the sliding window. At this point, another parameter which is the standard deviation threshold (θ_σ) should be determined. This parameter helps us identify the sliding windows in which a significant change in power value happened. The standard deviation threshold vector is defined as follows:

$$V_{th}[n] = \left\{ \begin{array}{cc} 0 & \text{if } \sigma_{n,n+w_s} \leq \theta_\sigma \\ 1 & \text{else} \end{array} \right\} \tag{3.11}$$

In V_{th} the indices with the value of 1, indicate the windows that contain an event. Figure 3.4 illustrates the performance of this event detection algorithm on a load profile.

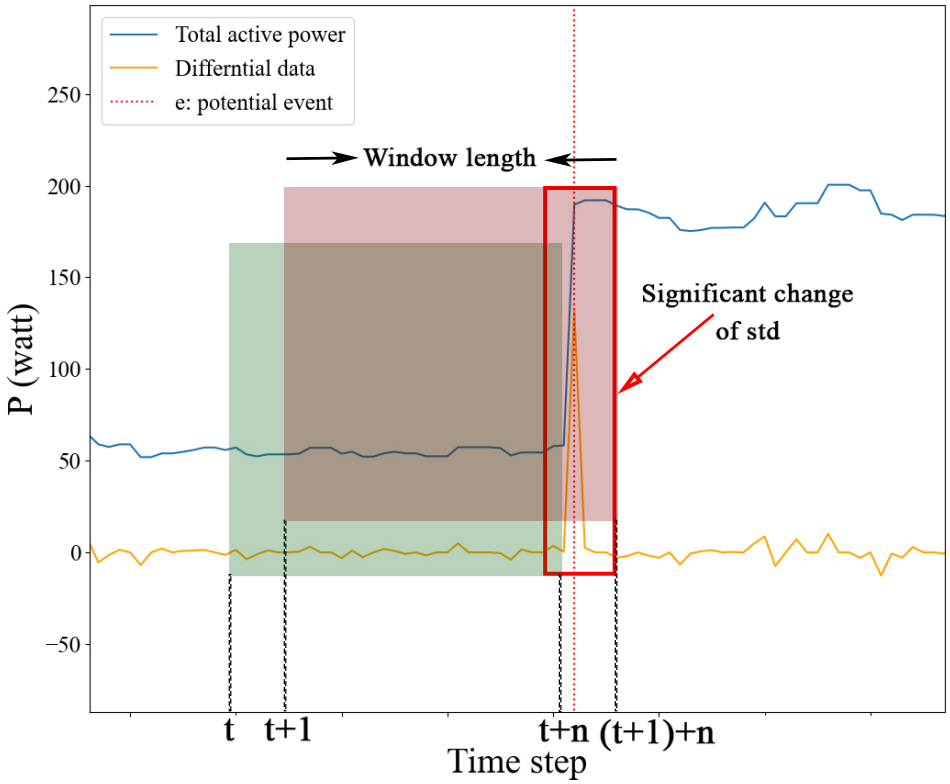

Figure 3.4 The performance of sliding window distribution change algorithm

So far, two of the most popular and effective event detection algorithms that are used in different studies, were explained. However, there exist small modifications for each of these

algorithms that improve their accuracy and performance. We wrap up this chapter by mentioning one of the most important modifications that are applicable to both algorithms introduced in this chapter. These algorithms indicate a vector of indices (timestamps) in which the events have happened while in reality, a load transition is not a moment, it is a period of transition from one steady state to another one. Thus, each algorithm, based on its parameters and sensitivity, identifies an event somewhere around that transition period (either inside or outside). In theory, one may be willing to identify an event at the exact beginning moment, i.e., the rising edge of the transition, however, in NILM, detecting an event close to its actual timestamp is still great for the disaggregation purpose. Thus, in the evaluation step, a margin of acceptance for the identified events can be considered. It means, if a detected event is in a time interval before or after the actual (ground truth) event, it is considered as a true positive. It can be formulated as follows:

$$\text{Detected event} = \left\{ \begin{array}{cc} TP & \text{if } |t_{\text{ground truth}} - t_{\text{detected}}| \leq \theta_t \\ FP & \text{else} \end{array} \right\} \tag{3.12}$$

The second important modification is the elimination of the detected events that are too close to each other. In NILM, a period of steady state is expected after a transition, however, that is not always the case. There are moments in load profiles when a transition contains an overshoot followed by some fluctuation (like in the turn-ON moment of an AC). In these cases, the algorithms identify more than one event around that period of transition which is not true. In fact, it is not very common for residential consumers to turn on several appliances at the same time, thus this problem should be handled. To overcome this issue usually a uniqueness threshold (θ_u) around every detected event is defined. It means that when an event is detected somewhere in the load profile, no other event is considered in an interval around it. Thus any other event detected at a distance less than θ_u to the main event, will be removed from the events vector. In this thesis, we consider both θ_u and θ_t equal to 4.

In the next chapter, we explain the fundamentals of Reinforcement Learning (RL) which is the core algorithm in the proposed method.

CHAPTER 4 MACHINE LEARNING

Machine learning (ML) is a topic of study focused on comprehending and developing learning methods that use data to enhance performance on a certain set of tasks. It is considered to be a subset of artificial intelligence. Without being expressly taught to do so, ML algorithms create a model using sample data, also referred to as training data, in order to make predictions or judgments. ML algorithms are utilized in a broad range of applications, including computer vision, natural language processing, speech recognition, email filtering, medicine, and fraud detection when it is challenging or impractical to create traditional algorithms that can accomplish the required tasks. Depending on the type of signal or feedback that is provided to the learning system, machine learning algorithms are generally categorized into three major groups that correlate to learning paradigms. Supervised learning, unsupervised learning, and reinforcement learning are the three main groups of ML algorithms that are explained briefly in what follows:

4.1 Supervised learning

Using inputs and outputs from a collection of data, supervised learning algorithms create a mathematical model of the data. The set of training examples makes up the data, which is referred to as training data. The intended output, also known as a supervisory signal, is present in each training example together with one or more inputs. In the mathematical model, the training data is represented by a matrix, and each training sample is represented by an array or vector, commonly referred to as a feature vector. A function that may be used to anticipate the output connected to fresh inputs is learned using supervised learning algorithms through iterative optimization of an objective function. The algorithm will be able to accurately predict the result for inputs that were not included in the training data thanks to an optimal function. An algorithm is considered to have learned to accomplish a task when, over time, the accuracy of its outputs or predictions increases. Figure 4.1 illustrates the work cycle of a supervised algorithm.

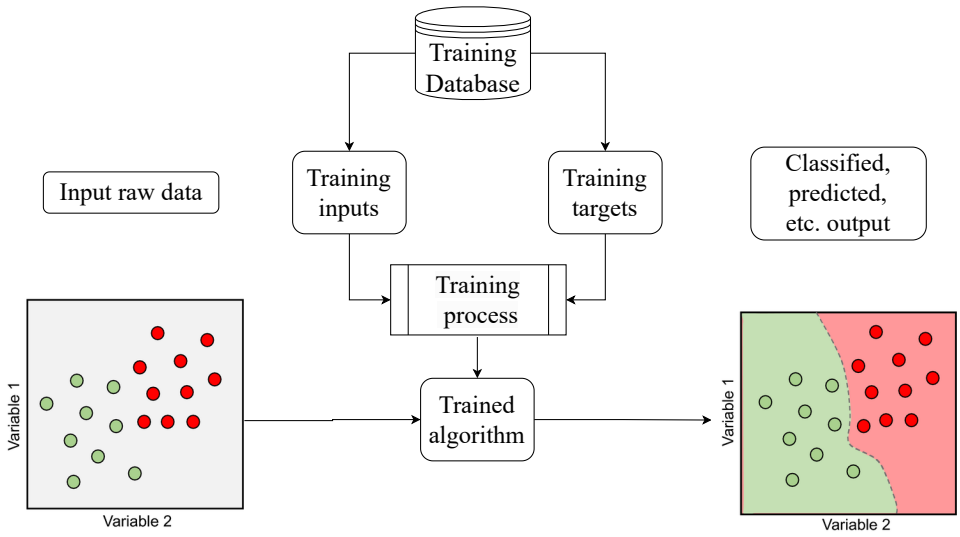

Figure 4.1 A supervised learning algorithm workflow

4.2 Unsupervised learning

Unsupervised learning algorithms analyze a collection of input-only data to identify patterns, such as grouping or clustering of data points. Therefore, test data that has not been labeled, classed, or categorized is used to train the algorithms. Unsupervised learning algorithms locate similarities in the data and act based on the existence or absence of such commonalities in each new piece of data, as opposed to reacting to feedback. Unsupervised learning is widely used in the area of density estimation in statistics, which includes determining the probability density function. Unsupervised learning, however, also applies to fields that summarise and describe data characteristics. When doing a cluster analysis, a set of data is divided into smaller groups, or clusters, where observations from the same cluster are similar based on one or more predetermined criteria, while observations from other clusters are not. Different clustering methods base their assumptions on different aspects of the data structure, which is frequently determined by some kind of similarity metric and assessed, for instance, by internal compactness, or the similarity between cluster members, and separation, or the distance between clusters. Other approaches rely on graph connectedness and estimated densities. Figure 4.2 illustrates how an unsupervised algorithm works.

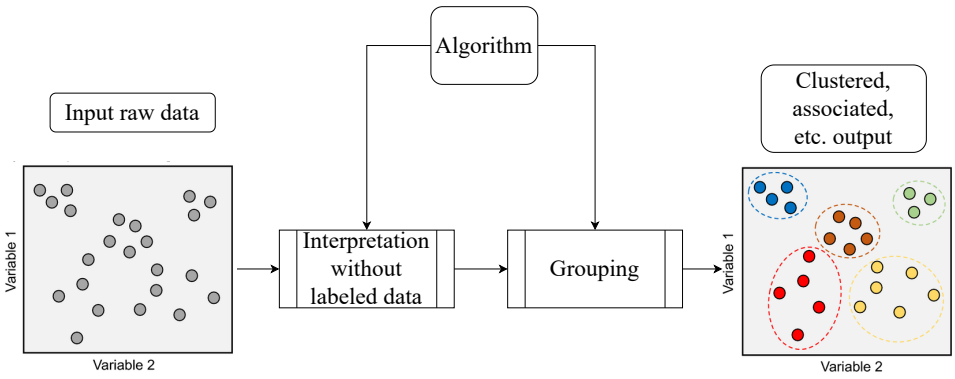

Figure 4.2 An unsupervised learning algorithm workflow

4.3 Reinforcement Learning

The field of machine learning known as reinforcement learning (RL) studies how intelligent agents should behave in a given environment to maximize the concept of cumulative reward. Along with supervised learning and unsupervised learning, reinforcement learning is one of the three fundamental machine learning paradigms.

Reinforcement learning differs from supervised learning in that it does not need the presence of labeled input/output pairings or the explicit correction of sub-optimal behaviors. Instead, the emphasis is on striking a balance between exploitation (of current knowledge) and exploration (of undiscovered terrain).

Since many reinforcement learning methods for this situation involve dynamic programming approaches, the environment is generally expressed as a Markov Decision Process (MDP). The fundamental distinction between traditional dynamic programming techniques and reinforcement learning algorithms is that the latter does not require an understanding of a precise mathematical representation of the MDP and is more suitable for big MDPs than precise techniques are. In many fields, including game theory, control theory, swarm intelligence, information theory, simulation-based optimization, multi-agent systems, operations research, and statistics, reinforcement learning is researched due to its universality. Reinforcement learning is also known as neuro-dynamic programming or approximation dynamic programming in the operations research and control literature. Due to the importance and the key role of RL in this research, RL concepts are explained more in-depth compared to supervised and unsupervised methods here.

4.3.1 RL working principles

Every RL algorithm has two main parts, the agent and the environment. These two entities communicate through three signals, i.e., action (A), state (S), and reward (R). Thus, a basic RL problem can be modeled as a Markov decision problem (MDP):

- S, is a set of environment and agent states;
- A, is a set of actions for the agent;
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the likelihood (at time t) that state s will change to state s' with action a
- $R_a(s, s')$ is the immediate reward of the change from state s to state s' under action a

Figure 4.3 shows how the agent and the environment communicate in this algorithm.

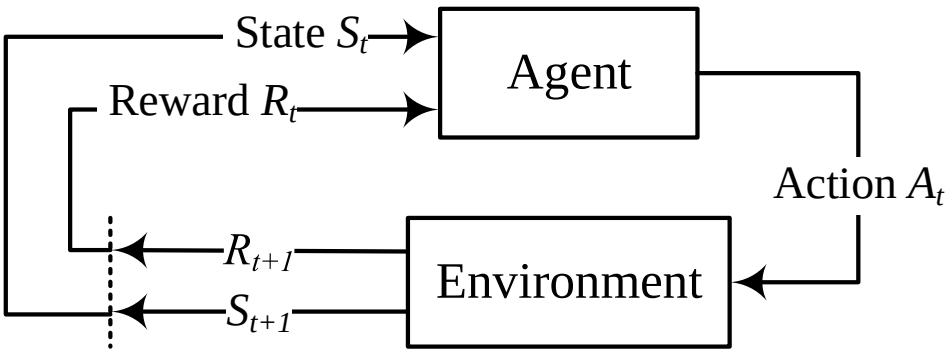

Figure 4.3 The reinforcement learning algorithm working cycle

In order to optimize the reward function or other user-provided reinforcement signal that builds up from the immediate rewards, the agent uses reinforcement learning to develop an optimum. Similar processes seem to take place in animal psychology. Biological brains, for instance, are designed to perceive signals like pain and hunger as negative reinforcements and pleasure and food intake as positive reinforcements. Animals can learn to adopt behaviors that maximize these rewards in certain situations. This shows that animals are able to learn through reinforcement.

Basic reinforcement learning agents engage with their environment in discrete time steps. When time t occurs, the agent is given the current state s_t and reward r_t. From the list of possible actions, it selects one and sends it to the environment in the form of action a_t. The environment changes to a new state s_{t+1}, and the reward r_{t+1} connected to the

transition (s_t, a_t, s_{t+1}) is established. A reinforcement learning agent seeks to discover a policy $\pi : \mathbf{A} \times \mathbf{S} \rightarrow [0, 1]$, $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$ that maximizes the predicted cumulative reward.

If the agent directly observes the present environment state, then the issue is said to have complete observability when it is formulated as an MDP. The agent is said to have partial observability if it only has access to a portion of the states or if the observed states are tainted by noise. In this case, the problem must be formalized as a partially observable MDP. The set of activities that the agent is capable of performing can be limited in both situations. For instance, if the current value of an account balance is 3, and the state transition tries to lower the value by 4, the transition will not be permitted. The state of an account balance might be restricted to be positive.

The concept of regret is introduced when the agent's performance is contrasted with that of an agent that behaves optimally. The agent must consider the long-term effects of its actions (i.e., maximize future revenue) in order to behave almost optimally, even when the immediate reward associated with this may be unfavorable.

As a result, problems, where there is a trade-off between long-term and short-term rewards, are particularly well suited for reinforcement learning. It has been used to solve a variety of issues, including some of the most complex games, e.g., Go (AlphaGo) [43].

The utilization of samples to maximize performance and the use of function approximation to handle large environments are two factors that contribute to the effectiveness of reinforcement learning. These two essential elements enable the application of reinforcement learning in large environments for the following cases:

- When there is a model but the analytical solution is not available;
- Only a simulation of the model is available;
- There is no information about the environment and the only way to collect data about that is to interact with the environment.

Since there is some kind of model accessible, the first two types of problems may be classified as planning issues, whilst the final issue could be classified as an actual learning issue. Both planning issues are, however, transformed into machine learning issues via reinforcement learning. Knowing the RL working principles, it is useful to briefly explain some terms and concepts that are usually used in most RL algorithms.

4.3.2 Exploration vs. exploitation

An RL agent should have a method to choose between exploration or exploitation at every action-choosing moment. The agent explores new actions in a given state with the hope to maximize the cumulative reward. On the other hand, the agent can choose short-sightedly by taking an action that yields the maximum reward at that time step regardless of cumulative future reward. Exploration methods can be simple or complex, static or dynamic. One of the most common exploration methods is ϵ-greedy where $0 < \epsilon < 1$ is a variable that regulates how much is spent on exploration vs exploitation. The agent picks exploitation with probability $1 - \epsilon$ and does the action that it thinks will have the best long-term consequence. Instead, when probability ϵ is used, exploration is selected and the action is evenly picked at random. Parameter ϵ is typically a constant value but can be changed either adaptively based on heuristics or according to a logic (making the agent explore gradually less).

4.3.3 Policy and optimization

A policy is formally a mapping between states and the probability of choosing each possible action. If the agent is following policy π, then $\pi(a|s)$ at time t is the probability of taking the action $A_t = a$ if $S_t = s$. In fact, the agent decides based on its decision policy, π, and its exploration-exploitation policy. The goal of both is to maximize the cumulative reward in the future. There are also deterministic policies in some RL algorithms.

$$\begin{aligned} \pi &: \mathbf{A} \times \mathbf{S} \to [0, 1] \\ \pi(a, s) &= \Pr(A_t = a \mid S_t = s) \end{aligned} \tag{4.1}$$

Knowing the meaning of the policy, we can define the next necessary concept for training an RL agent which is the value function (or state-value function). The value function of a state s successively following policy π, denoted as $v_\pi(s)$, is the expected return starting with state s, i.e., $s_0 = s$. In other words, it means "how good" it is to be in a given state. v_π can be defined formally as follows:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ for all } s \in S \tag{4.2}$$

where the variable G is defined as the return, i.e., the sum of the future discounted rewards from the time t to the last time step. r_t is the reward at time step t and $\gamma \in [0, 1)$ is the

discount factor which makes the reward of distant time steps weigh less than immediate future rewards for the agent. It is worth mentioning that the value of the terminal state is 0 because no more future rewards exist in that time step. The algorithm should find the policy that maximizes the expected reward, i.e., an algorithm that knows which state leads to the highest cumulative reward over the course of the next steps. Almost equivalently, the action-value function can be defined as a function for a given state (state action-value function) that determines how good each action in a given state is in order to make the cumulative future reward maximum. It can be formulated as below:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[\, G_t \mid S_t \;=\; s, A_t \;=\; a \,\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{4.3}$$

The RL algorithm tries to find a policy to maximize the cumulative future reward (expected return). Since $v_\pi(s)$ can be written recursively, the algorithm can interact with the environment, experience different states, take different actions, and see the results. Consequently, the algorithm can compare the reward of every action in different states and step by step optimize its policy. There are different methods in order to recursively evaluate and optimize the policy. For example in stationary policies, the agent takes an action based on the most recent state that it has visited, or in deterministic policies, the agent takes an action based on a rule that only checks the current state for decision-making. In some methods, the agent creates a model (simulation) of the states and actions that it has never tried. Despite all this variety in methods, the recursive calculation of the state-value function leads to the Bellman equation as follows:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\left[\, G_t \mid S_t \;=\; s \,\right] \\ &= \mathbb{E}_\pi\left[\, R_{t+1} + \gamma G_{t+1} \mid S_t \;=\; s \,\right] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \, \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']]) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \, v_\pi(s')], \text{ for all } s \in S \end{aligned} \tag{4.4}$$

Solving an RL problem means finding a policy that gains the maximum possible reward over the long run. Policy π is considered better than or equal to π' if π's expected return is greater or equal to that of π''s in all states. In other words, $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$. There is always at least one policy to have this condition. This policy is called an optimal policy and it is denoted as π_*. Subsequently, the state-value function and state-action value function corresponding to the optimal policy are also the optimal functions represented as follows:

$$v_*(s) \doteq \max_{\pi} v_\pi(s), \text{ for all } s \in S \tag{4.5}$$

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a), \text{ for all } s \in S \text{ and } a \in A \tag{4.6}$$

Having the Bellman equation, the algorithm can calculate state values recursively and tries to improve the state-value function until it converges to v_*. There are several methods in the RL field, with which the state-value function is optimized. In Monte Carlo methods, a repetitive cycle of policy evaluation and policy improvement tries to find a better policy with every iteration and replaces the old policy with the more efficient newly found one. To overcome Monte Carlo method inefficiencies, Temporal Difference methods were applied to the RL optimization process in which the policy changes partially before being completely replaced by another policy. There exist many other methods for this step. As it is a very extensive topic, it is out of context to go any further, and we focus on our proposed method and RL optimization in the next section.

CHAPTER 5 REINFORCEMENT-LEARNING-BASED NILM

In this chapter, the proposed solution is thoroughly explained. First, the problem to which we seek a solution is defined to make the goal clear. Since there are existing event detection algorithms for the NILM purpose, it is crucial to explain what difference our proposed solution makes. As the data is the most important part of any artificial intelligence algorithm, in the second section, the data structure is explained. Then, we discuss how a NILM event detection algorithm can be made using RL architecture. Lastly, the Dual Replay Memory structure is explained which is a key element to improve the performance of any RL agent used for event detection purposes.

5.1 Problem definition

As discussed in Chapter 1, event detection is a core part of every NILM algorithm. There are several event detection methods that are being used in many studies or even real-world solutions. However, each of those methods works on a limited range of appliances, i.e. there is not one event detection method that works for all different load profiles. The following reasons may cause this problem:

5.1.1 Load type

In NILM literature, loads are divided into four different groups as follows:

- Type-I: simple ON/OFF appliances, e.g., a toaster, or a light bulb. These loads usually have a simple rectangular load signature.
- Type-II: Finite State Machines (FSM) are those appliances that go through a state transition. FSM cycles are repeated frequently, e.g., a refrigerator (OFF/ON/Defrost), or a cloth dryer (OFF/Heater+Motor/Motor only).
- Type-III: Continuously varying loads that can consume power in a continuous range and it is not limited to certain power levels, e.g., a dimmer light.
- Type-IV: Permanent consumer devices or Always-on devices that are hard-wired to the power source.

Figure 5.1 illustrates some of these load types. In the field of NILM, Type-I and Type-II appliances are of great importance due to their high power consumption level.

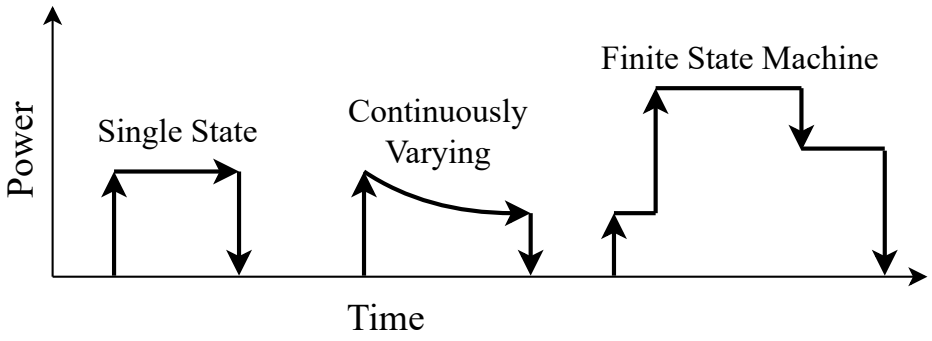

Figure 5.1 Different load types in the NILM literature

However, an ideal event detection algorithm should be able to identify all different load types. Type-I or simple ON/OFF appliances are the easiest ones to identify because of their sharp rising and falling edges. Type-II or FSMs are more complex as they have more than one state and some of these states include numerous fluctuations which makes it really hard for any algorithm to identify an actual event. Figure 5.2 illustrates a cycle of a washing machine in which many sharp fluctuations are happening. Considering the fact that each cycle of an FSM has its own characteristic, FSM event detection becomes even more complex.

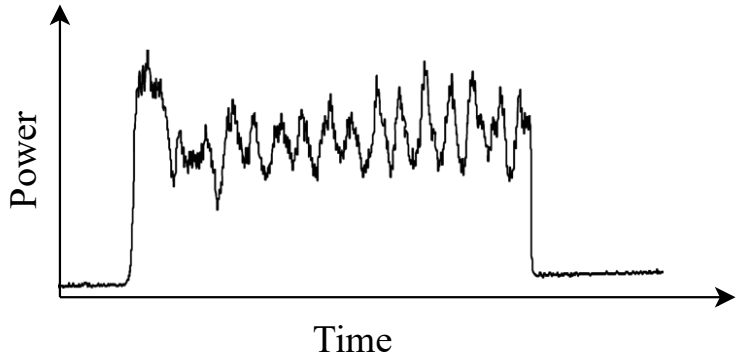

Figure 5.2 Load signature of a washing machine

Type-III or variable loads are even harder to identify than the previous type. When a load does not have a specific ON or OFF moment it is very hard to identify or classify it. On the other hand, the power consumption level of these appliances can change at any time which may be mistaken for an actual event. In the last group of loads, type-IV, no turn-ON or turn-OFF moment can be observed as they are always connected to the power source. Thus, no event detection function can identify them. However, their power consumption level can be reported as the base load of the targeted house. Because there are not many important

appliances in the Type-III and Type-IV loads and their power consumption is low, not many studies have focused on them, however, in the near future Electric Vehicles (EVs), which are considered as continuously varying loads, will be more prevalent in households and event detection algorithms have to be able to identify this important group of loads too.

5.1.2 The nature of the algorithm

In Chapter 3, three main categories of event detection algorithms are defined. However, there exist less-known methods. Each type of event detection algorithm works on a particular basis that only applies to specific appliances or load types. For example, one event detection algorithm may use Fourier Transform to identify a load event by analyzing its current harmonics. This type of event detection may not be able to identify load events properly as the harmonics in the transient state in some appliances are different than that of their steady state while in some other appliances, the turn-ON moment harmonics are the same as that of the steady state. As another example, in probabilistic methods, the algorithm is searching for a significant change of value distribution in active power, however, modern electronic appliances do not have a significant power consumption level compared to the main appliances in the home (e.g., fridge), thus, the power values distribution in the load profile does not change notably in order to be discovered by the probabilistic event detection algorithms.

5.1.3 Parameter optimization

Most event detection algorithms have some parameters that need to be adjusted or optimized in order to detect load events accurately. For example, consider the log-likelihood ratio detector that was explained in Chapter 3. This event detection algorithm needs five parameters to operate properly. Power threshold (θ_P), pre-window (w_0), post-window (w_1), voting threshold (θ_V), and voting window (w_V) optimal values should be calculated by parameter sweeping method and set in the algorithm. The optimal values of each of these parameters are dependent on the load profile that the algorithm is scanning for event detection. In this case, the power threshold θ_P determines the minimum power level that should be observable by the LLR algorithm. If it is set too low, the LLR algorithm perceives many small fluctuations as possible events. In fact, it increases the sensitivity of the algorithm. It can be a good attribute because the algorithm identifies small appliance (low power) events. However, with a low power threshold, the algorithm may confuse a small power fluctuation with an actual load event which will lead to increased False Positive (FP) detected events. Other parameters need to be adjusted too. As another example, a short pre-window or post-window may make the algorithm very fast, however, it disables the ability to find significant

changes in the power values distribution by the algorithm, and as a result, the number of True Positives (TP) events will be reduced.

Almost all event detection algorithms suffer from the parameter optimization problem. One optimized algorithm can not be applied to all different load profiles with different characteristics, because there are various power levels, usage frequencies, and appliances type in each load profile.

5.1.4 Input requirements

The chosen electric signal and its sampling frequency are the most important inputs for the event detection algorithm. Although the electric signals (P, Q, I, and V) reflect significant changes in power consumption almost simultaneously, their levels, fluctuations, and patterns differ. One event detection may be designed to identify load events by comparing current harmonics while another event detection algorithm finds load events by identifying active power changes. The minimum frequency and the electric signal of both these algorithms are different. This fact makes it harder to use these algorithms alternatively, i.e., you should choose the algorithm that matches your available data in the grid or provided by your sampling device. One can conclude that despite the fact that all these signals contain the same information about the load events, the same input can not be used in all event detection algorithms. Each algorithm should be provided with its specific electric signal and the required frequency in order to work properly [44].

A versatile event detection algorithm is the core part of any NILM algorithm. Without identifying load events, it is hard for the rest of the parts in a NILM algorithm to fulfill their task properly. Having the above-mentioned problems of event detection algorithms in mind, helps us understand why it is not possible to use one event detection algorithm for all load profiles, despite the fact that an event detection algorithm may work perfectly for a specific group of appliances in a load profile. One may suggest designing a separate event detection algorithm for each group of appliances to have a comprehensive event detection process. This is not an optimal solution as there is no way to handle the contradiction between different event detection algorithms' output at some time steps.

Each event detection algorithm has knowledge (a model) about identifying specific load events, even if it is not perfect. We propose that there is a way to gather all these sets of detection processes in one single model that can perform the task of all those individual event detection algorithms. Thus, our goal is to propose an architecture and an algorithm that can learn the detection process of all other event detection methods regardless of their working principles. This new model is supposed to use the input data that are available in the grid

and it should not be restricted with a specific electric signal or a sampling frequency range.

A tangible example is helpful to clarify the situation. Imagine there are two professors in a department. Professor X can answer 90% of questions in his field A, and Professor Y can answer 85% of questions about field B. Assume there is a student who knows nothing about either topic A or B. This student is supposed to take an exam which contains 50% of topic A and the rest is of topic B. If each professor is asked to separately take this exam, they will get a high score in their specific 50% of questions but they will leave the rest of the questions blank. Now, if professors are asked to teach the mentioned student their own topics, and the student learns even 60% of each topic, the final result of the exam would be much better than the final score of each teacher.

This analogy makes it clear why it is important to aggregate the knowledge of many event detection algorithms instead of trying to make them perfect separately. An event detection algorithm that works acceptably on all different load profiles is much more useful for the NILM industry than an algorithm that works perfectly on a limited range of load profiles.

In the next sections, it is explained how the process of knowledge aggregation of different event detection algorithms can be addressed by our purposed architecture and algorithm.

5.2 Overview of the proposed solution

In Chapter 4 we explained what are the main parts of a reinforcement learning problem. As it is pointed out before, the best analogy for an RL problem is a video game. In video games, there is an agent (player) who has limited moves. These moves and actions may be combined sometimes, however, there is nothing to do more than those allowed moves. The agent should take those actions to get points in the game environment which enforces certain rules. The total score is observable to the agent (player) and the agent wants to maximize it. Having this background in mind, it is easier to explain how a NILM event detection task can be translated into an RL problem.

Figure 5.3 illustrates the modular arrangement of an RLNILM problem. In the first layer, we observe the environment and the RLNILM agent. The environment is supposed to provide the RLNILM agent with the state at the current time step and the reward for the previous action of the agent. In our case, the states are all available signals that are being sampled in the targeted unit. The RLNILM agent decides to take an action based on the state received from the environment and the reward it gets from the rewarding function. The RLNILM agent creates an output which is in fact a signal that indicates if an event has occurred at the current time step or not (binary). This output will then be used as the feedback signal for

the reward function. The reward function is responsible to judge the RLNILM agent action in the previous state. The reference for this judgment is provided by the feedback optimizer module which processes the outputs of other event detection algorithms embedded in the feedback system, for the same state. If the RLNILM agent's decisions match that of the feedback system, the reward function encourages the RLNILM agent by giving it a positive reward, otherwise, the reward function punishes the agent with negative scores. Even though the RLNILM agent does not know anything about what it is doing at first, it will learn to take actions that match the collective knowledge of all event detection algorithms in the feedback system in order to gain the highest cumulative reward.

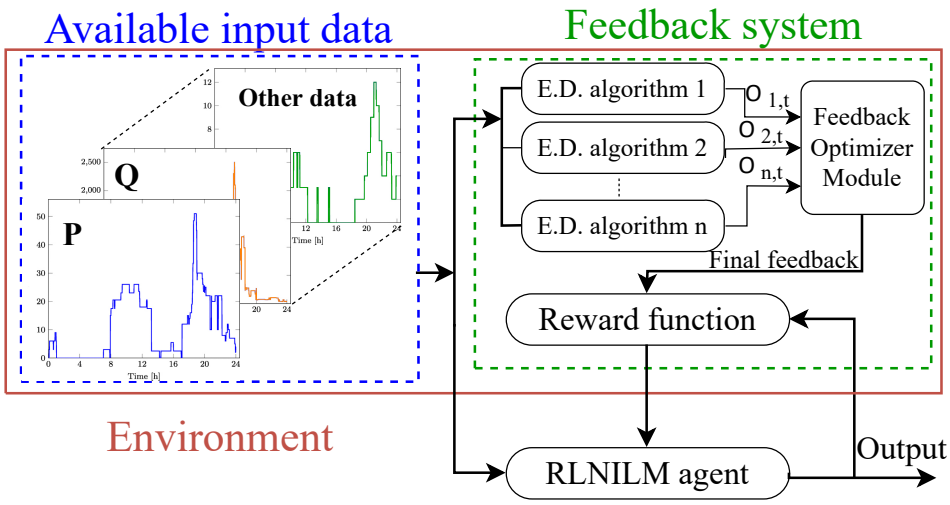

Figure 5.3 The modular view of the proposed RLNILM arrangement

Now, the game in which the RLNILM agent is playing its role is clear to us. The RLNILM agent does not know anything about the working principles of NILM event detection algorithms embedded in the feedback system. It learns their detection processes by observing the reward of its actions. The RLNILM does not require the feedback system to have any specific combination of event detection algorithms. It can work with any number of event detection algorithms in its feedback system. Equivalently, the RLNILM agent does not force any requirements to input data. As long as the embedded event detection algorithms are able to function with the provided input data, the RLNILM agent tries to extract their knowledge in the best possible way. It is obvious that lowering the input data quality may impact the embedded event detection algorithms' performance adversely, however, it is important to note that the RLNILM agent still tries to learn the most out of that situation. In other words, the flexibility of the RLNILM agent in learning the collective knowledge of other algorithms is what makes it very useful. The next sections go deeper into the details of each part of the

RLNILM structure.

5.3 Environment

The environment is the biggest part in Figure 5.3. In the general form of RL, it is responsible for creating a sequence of states and rewards based on the agent's action at every time step as below:

$$S \rightarrow A \rightarrow R \rightarrow S \rightarrow A \rightarrow R \rightarrow \cdots \tag{5.1}$$

where S, A, and R stand for state, action, and reward respectively. In our proposed method, the environment contains one extra part which is the feedback system. The existing event detection algorithms are considered a part of the environment with which the RLNILM agent is interacting.

The input data is simultaneously presented to both the feedback system and the RLNILM agent. One state is generated at every time step. The feedback system is responsible to create a feedback signal for rewarding function. As there may be several event detection algorithms embedded in the feedback system, a feedback optimizer module was designed to unify and optimize all the outputs of the event detection algorithms and provide the reward function with a final feedback signal which is used as a reference to judge the actions of the RLNILM agent based on the given state at that time step. The reward function reads the output of the RLNILM agent (which is the final output of the RLNILM solution) and compares it with the final feedback signal from the feedback optimizer module. The reward function then encourages or punishes the RLNILM agent to repeat or avoid such a decision in the given state, by positive or negative numeric values. In the following sections, the details of each part of the environment are explained.

5.3.1 Input data

As it is mentioned in Chapter 2, the iAWE dataset is used as the foundation of the data generation algorithm in our proposed method. The iAWE dataset is a real-world dataset sampled at 1 Hz frequency in a house in India. It contains different electric signals including I, P, and Q. It contains both appliance level and aggregated (mains) consumption measurements. However, it lacks load events data. In fact, it is very rare in publicly available NILM datasets to have the indicator of ON/OFF moments of the monitored appliances. Figure 5.4 illustrates the measurement of active and reactive power for some of the monitored appliances in this dataset.

It is crucial in event detection studies, to know when a load has turned ON or OFF, because

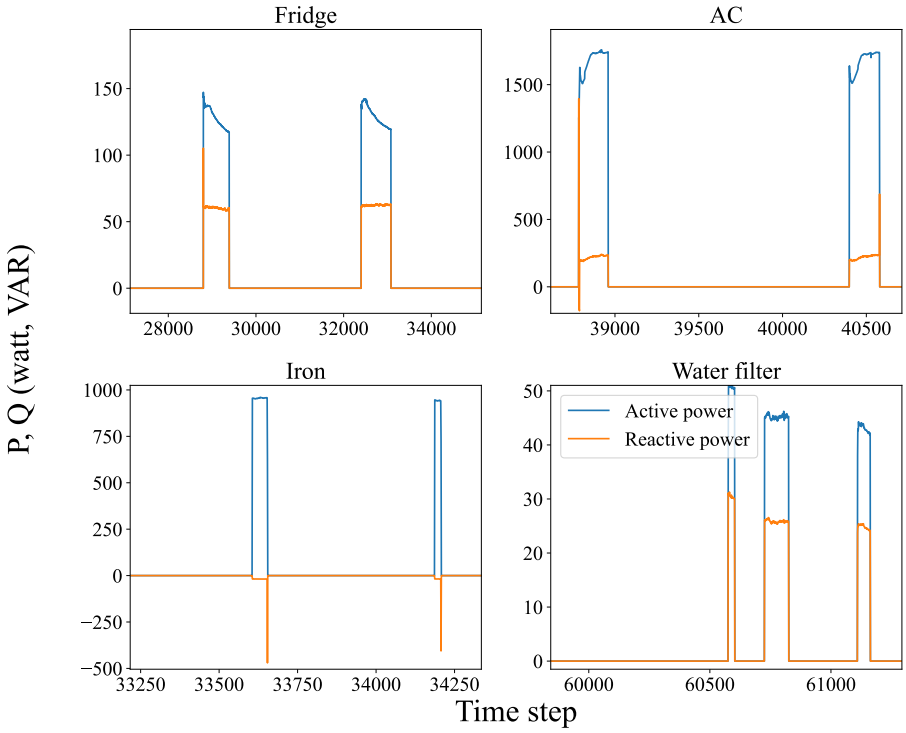

Figure 5.4 P and Q of different appliances in iAWE dataset

without that information it is almost impossible to evaluate and improve the performance of the designed event detection algorithm. Another issue that arises with the input data is the fact that many machine learning methods, especially RL methods, are dependent on a high number of repetitive cycles for their learning process which demands a huge number of samples. In all real-world NILM datasets, we can hardly find a dataset that continuously monitors each appliance consumption at 1 Hz frequency or higher, for a sufficient time interval. Thus, it was not possible to train an RL algorithm with that limited number of load events in the available public datasets. In order to overcome these barriers, we decided to generate an artificial dataset using the load signatures in the iAWE dataset. In this manner, labeled load profiles can be produced as long as it is needed for training the RLNILM algorithm. Thus, all the load signatures (data between each ON and OFF moment) of different appliances in the iAWE dataset are extracted and saved as CSV files. Then, a sequence generator algorithm rearranged all the load signatures with a normal household consumption pattern and marked the turn-ON and turn-OFF moment of each load signature with its specific code from Table 2.1 in the sequence key array. Lastly, we added the Gaussian noise with the mean and standard deviation of the iAWE dataset to the final sequence to make it

as realistic as possible. Figure 5.5 shows an artificially-made load profile and its labels look like.

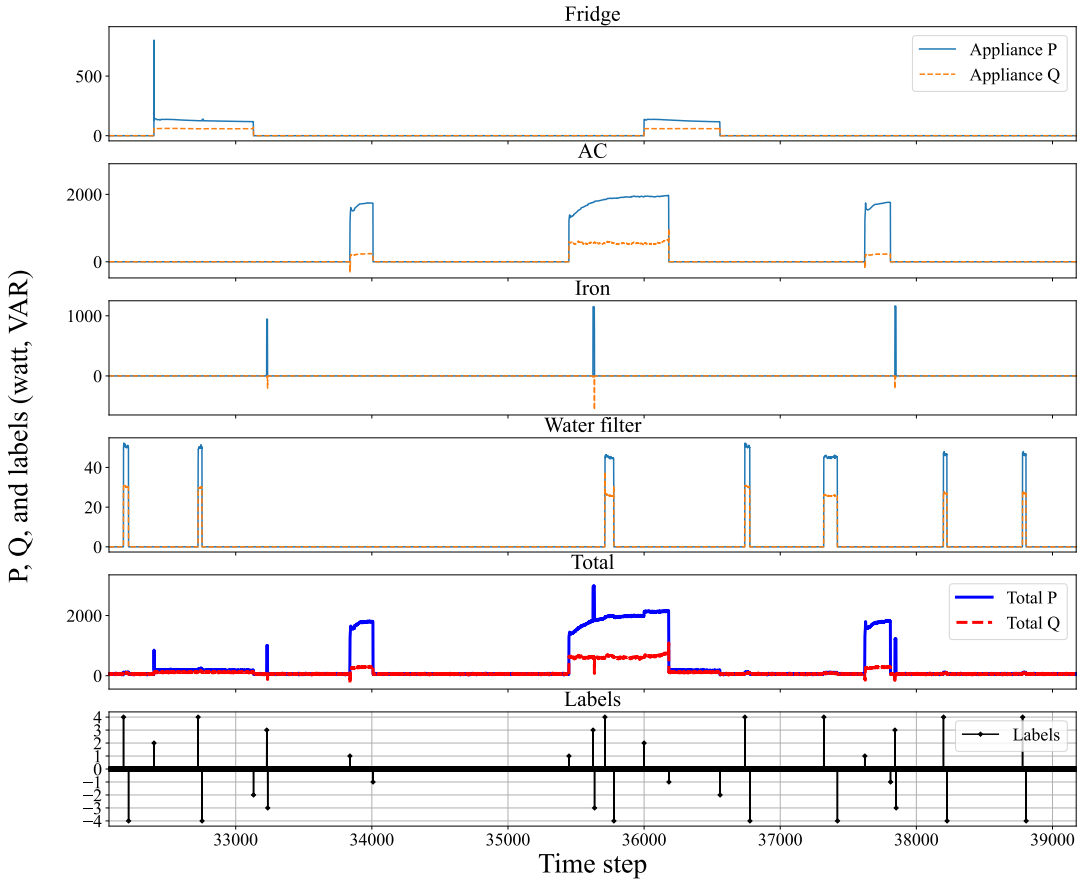

Figure 5.5 A randomly-generated sequence from the iAWE dataset with labels (labels are scaled from -4 to 4)

Now, the environment can generate unlimited new states with real-world load signatures for the training and evaluation purpose of the RLNILM method.

Earlier in this chapter in Section 5.1 we pointed out some of the problems with the use of existing event detection algorithms. One of the most important barriers to using them is their input data requirements. As it is explained before, different event detection and NILM algorithms, need different inputs. This difference may be in the type of electrical signal or its sampling frequency. Therefore, the prerequisite for employing an event detection algorithm in a NILM solution is to provide that algorithm with its needed input, which may not be

readily accessible in the grid. However, the RLNILM event detection method is more flexible in this sense. As it can be observed in Figure 5.3, we can add all available input streams, including but not limited to I, V, P, Q, room occupancy, and weather. The event detection algorithms embedded in the feedback system may or may not be able to work efficiently with them. However, the RLNILM agent does not enforce any requirement on the quality of those input streams. It is capable of using and learning from all the information hidden in input signals with the instruction of the reward function. A tangible example makes it easy to understand the advantage of the RLNILM method. Consider an event detection algorithm, ED1. ED1 is a probabilistic LLR algorithm that needs a high-frequency-sampled active power signal. Assume we want to use this algorithm in a house that provides a lower frequency P signal but instead, it has room occupancy data. Fewer detected events can be expected for ED1 (needs high-frequency P but gets low-frequency P). With a lower sampling frequency, ED1 which depends on sudden changes of P to identify a load event, will have difficulties to identify nearly simultaneous events because of the low sampling rate. Thus, existing event detection algorithms are heavily dependent on their input signals. Assume that ED1 is used in the RLNILM structure. ED1 is still reading low-frequency P as its input, while RLNILM is reading both P and room occupancy data. There is important information in the room occupancy signal, because specific room occupation may be correlated with the use of a specific appliance. ED1 is deprived of this valuable information, thus, when the house owner enters his TV room to watch TV as always, ED1 may miss the TV turn-ON event due to simultaneous events somewhere else in the house, but RLNILM has learned that occupying the TV room is mostly involved with TV turn ON event. This is a very basic example of how RLNILM can use and co-relate all available data in the grid (or target monitored unit) while other event detection algorithms will be impaired by any change in their input streams.

5.3.2 States

In RL literature, the problem values that describe the current configuration of the environment are called states. The agent takes action based on the given state at every time step. A limited sequence of states is called an episode. To better understand the meaning of an episode in RL problems, imagine a simple video game. In a video game, the concept of an episode is the same as the concept of a level. The player should replay the same episode or proceeds to a new level if:

- The player reaches a specific score (wins)
- The player takes one or a series of bad actions (Game over)

- The player runs out of time or allowed moves

In many RL problems, the environment has the above-mentioned rules. Thus, the RL agent plays an episode several times until it learns how to win that level. A good example of such an episodic problem is the load flow problem in power grids. The agent does the load distribution over and over until it learns to satisfy all the grid constraints. On the other hand, in RL, problems may exist that can not be divided into different episodes, i.e., they are one long episode that the agent should deal with. NILM event detection is of this type. Of course, a load sequence can be divided into smaller pieces to become less computational-intensive, however, there is no winning, losing, minimum, or maximum score to finish the episode. Event detection is a continuous task and the goal of the RLNILM agent is to identify all load events correctly.

As it is discussed in Section 5.3.1, we used P, Q, and I as input streams. the values of these three variables determine the configuration of the environment for the RLNILM agent. Thus, a sequence of states generated by the environment can be modeled as follows:

$$s_0 \rightarrow \cdots \rightarrow s_{t-1} \rightarrow s_t \rightarrow s_{t+1} \rightarrow \cdots \rightarrow s_n \tag{5.2}$$

where s_t is the state at time step t and n is the maximum number of states in one episode. It can be further detailed as:

$$(P_{0,w_i}, Q_{0,w_i}, I_{0,w_i}) \rightarrow \cdots \rightarrow (P_{t,w_i}, Q_{t,w_i}, I_{t,w_i}) \rightarrow \cdots \rightarrow (P_{n,w_i}, Q_{n,w_i}, I_{n,w_i}) \tag{5.3}$$

where w_i is the size of the window that rolls over each of the input streams. In existing public NILM datasets, one can barely find a dataset with extra data about the weather or room occupancy, however, using that type of input requires longer sampling periods than what is available currently and it is not something that can be generated artificially as it should match the exact household real consumption behavior. For this reason, we pass up using such input in our experiment.

The sequence generator algorithm takes in inputs on desired distribution parameters of each appliance and creates a two-dimensional array with $3 \times w_i + 1$ columns and a row for each time step. To keep the algorithm memory-friendly, the length of each episode is limited to 24 hours. Considering the 1 Hz frequency of our dataset, each episode contains $24 \times 60 \times 60 = 86400$ states (rows). An example of a normalized load event of the sequence generator algorithm is shown in Figure 5.6. It is observed that the current signal contains less information (fluctuations and patterns) compared to active and reactive power signals

even on a normalized scale.

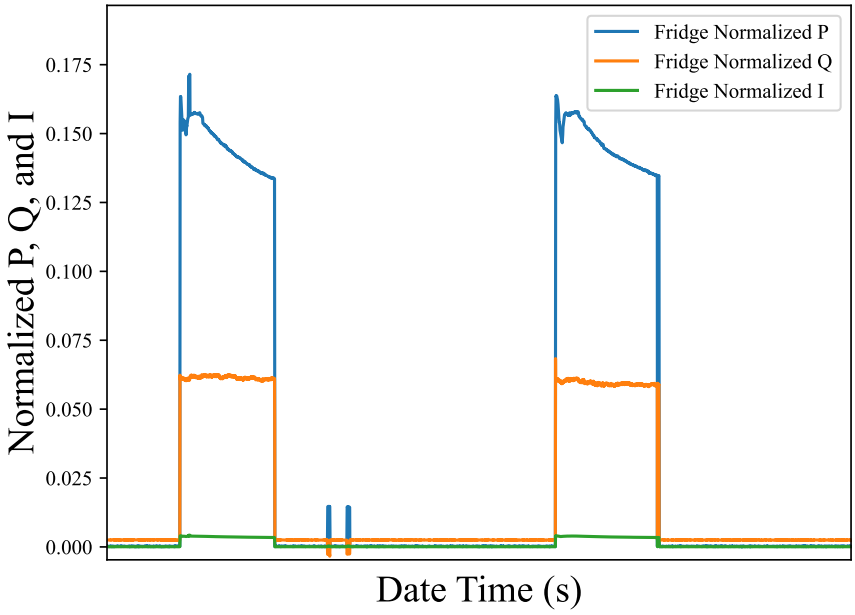

Figure 5.6 A fridge ON/OFF event normalized electric signals made by sequence generator

At every time step, the environment sends one state from the current episode to the RLNILM agent and the feedback system. When the RLNILM agent takes its action, then again, the environment sends the next state to both receivers (the RLNILM agent and the feedback system). One may wonder how these states that contain the w_i values of three different input streams may apply to other event detection algorithms that may need only one data point and not the entire rolling window of these input streams (e.g, Deep neural network event detectors can work with such input). The answer is that every event detection algorithm reads what it needs from the state window sent by the environment. For example, imagine there is an LLR event detection algorithm in our feedback system that needs a rolling window of 11 data points to identify if the 5^{th} point (the point in the middle of the window) is an event or not, and the environment is sending states with 20 data points width. Thus, the LLR algorithm reads the 11 points that it needs to investigate the occurrence of an event in the middle point. It should be noted that any event detection algorithm that needs a rolling window as its input, is in fact analyzing the middle point, so in this regard, there is no difference between sending one data point as the state or a rolling window as the state. Using rolling windows as the states enables us to use all different types of event detection algorithms in the feedback system.

The sequence generator algorithm puts the event type coding (according to Table 2.1) in the last column of the sequence array. This vector is called the key vector and it will be used for evaluation purposes.

Figure 5.7 illustrates how the input streams are turned into rolling windows of states for being analyzed by the feedback system and the RLNILM agent.

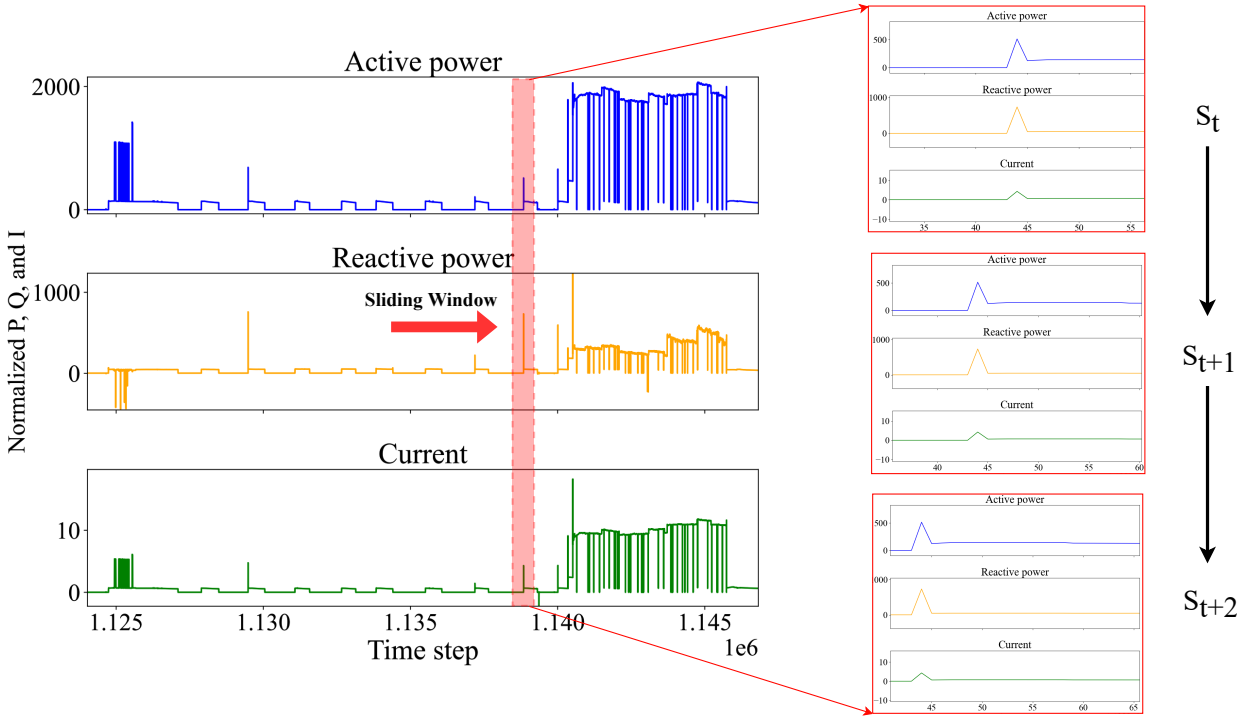

Figure 5.7 Input streams and sliding windows of states

5.3.3 Actions

Actions are a set of possible decisions that the agent can take in every state of the problem. In some cases, the agent can take all the possible actions in its action set A. However, there may be an environment and states in which it is not possible for the agent to take all actions, i.e., the set of possible actions changes with the state in which the agent is. Imagine a video game in which the player (the agent) should avoid hitting barriers in its path toward its destination. In the first level, the agent only can move left or right to avoid those barriers, but in the next level, it has the ability to shoot the barriers and destroy them. Thus, it means that based on the state s, the action set A of the agent has changed. However, in most cases, the action set of the agent is the same throughout the different states. Imagine an RL agent is supposed to find the optimal load flow for a given grid. Regardless of the changes in the

loads or transmission lines characteristics, the actions set of the agent would be the same in all states. In this case, the action set of the agent is choosing a number for each of the generation units. In some cases, even though the action set is limited and constant through all the states, the actions can be combined. For instance, if an agent is supposed to learn how to land an aircraft on the ground, it should be able to adjust the speed, and altitude while maintaining the aircraft balance at the same time. Thus, the actions may be chosen one at a time or many at the same time. They may be continuous or discreet. It all depends on the rules of the environment in which the agent is playing.

It is important to know how the action set of the proposed RLNILM event detection algorithm is defined. The goal of the RLNILM agent is to identify the time step in which an event happens. The environment presents each state to the RLNILM agent and it should decide if that state contains an event or not. Thus, it seems that a binary action set can appropriately represent this event detection's action set for the RLNILM agent.

$$a_t = \left\{ \begin{array}{cc} 1 & \text{if } s_t \text{ contains an event} \\ 0 & \text{else,} \end{array} \right\} \tag{5.4}$$

where t is the time step of interest and s_t is the state at that time step. This simple action set can fulfill its task (identifying the time step of the load events) properly, however, one may suggest a more complicated action set to merge two steps of a NILM algorithm, i.e., event detection and event classification. Recall from Chapter 1 that in many NILM algorithms, after the event detection step, the algorithm classifies (or clusters) the identified load events. Having the labels of different load events in our training data set (as in Table 2.1), it is possible to train a DNN-based event detection algorithm that not only identifies load events but can also determine which appliance ON or OFF event it belongs to. As a side experiment, we designed this algorithm and implemented it in our proposed method, however, to keep the entire experiment more general, it was preferred to use algorithms in the feedback system which can only identify load events and ignore the event classification task. Figure 5.8 illustrates how the proposed method was able to identify event types on the iAWE dataset. this action set can be itemized as below:

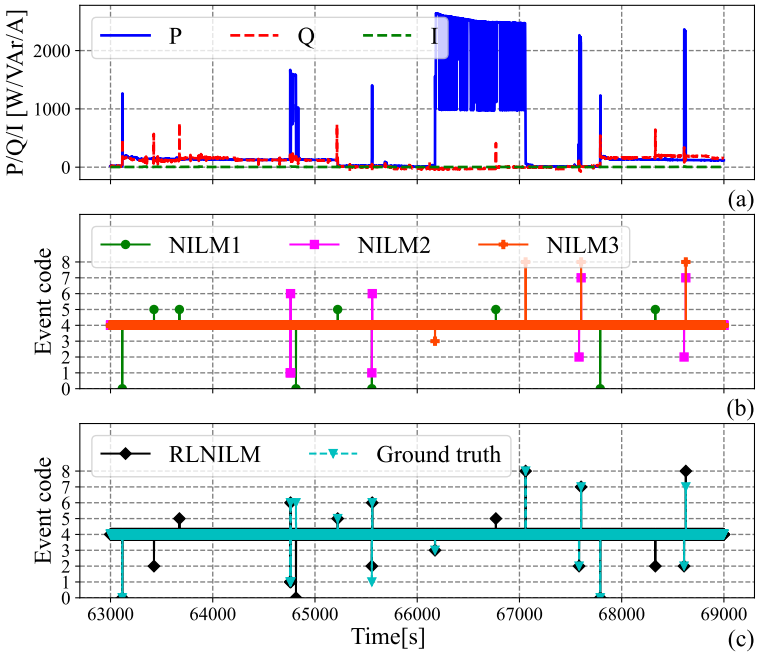

Figure 5.8 Load event type identification by DNN algorithms embedded in RLNILM architecture. a) input data streams b) embedded DNN algorithms output c) RLNILM output versus ground truth.

$$a_t = \left\{ \begin{array}{ll} 0 & \text{if } s_t \text{ contains AC turn ON event} \\ 1 & \text{if } s_t \text{ contains fridge turn ON event} \\ 2 & \text{if } s_t \text{ contains water filter turn ON event} \\ 3 & \text{if } s_t \text{ contains iron turn ON event} \\ 4 & \text{if } s_t \text{ contains no-event} \\ 5 & \text{if } s_t \text{ contains AC turn OFF event} \\ 6 & \text{if } s_t \text{ contains fridge turn OFF event} \\ 7 & \text{if } s_t \text{ contains water filter turn OFF event} \\ 8 & \text{if } s_t \text{ contains iron turn OFF event} \end{array} \right\} \tag{5.5}$$

Thus, there may be different action sets definable for the same task, based on what is expected to happen in the next steps of the process. As it is mentioned before, we choose the simpler actions set as in 5.4 to keep our proposed method as general as possible to be compatible with all different event detection methods which are supposed to be a part of another NILM algorithm. The RLNILM agent chooses an action from the action set A and presents it as the output for time step t at that iteration of the RLNILM algorithm. This output signal goes back to the reward function too, to be compared with the final feedback signal which

comes from the feedback optimizer module. In the next section, the role of the feedback system and how it helps the learning process of the RLNILM agent is explained in detail.

5.3.4 Feedback

The feedback system is a part of the environment that is responsible for extracting the knowledge of the existing event detection algorithms embedded in it. In previous chapters, it is mentioned that the reason behind using some of the existing event detection algorithms in the feedback system of our proposed method is extracting their knowledge, unifying and optimizing their output, and using them for the training purpose of the RLNILM agent. Consequently, a versatile RLNILM agent can be trained that is able to identify load events much better than any of those event detection algorithms used in its feedback system. We can add any event detection algorithm to our proposed architecture that works properly with the state information provided by the environment. They do not need to have the same methodology or architecture for event detection. They do not need to use the same input signals or the same sampling frequency. Thus, there is a wide range of options to choose the event detection algorithm to embed in the feedback system. Of course, each event detection algorithm works best when it is provided with its optimal input (frequency and signal type), however, they are still able to identify load events in the sub-optimal conditions. The goal is to improve their current performance by extracting and optimizing their knowledge via the RLNILM agent. In the NILM field, many event detection algorithms have been introduced. Two of them are explained in Chapter 3. Because of their popularity and performance, we decided to choose them as the test event detection algorithms to embed in the feedback system of our proposed algorithm. It is tried to set their parameters in a way that each of them is able to identify a specific group of events in our test load profiles. As the final goal, the RLNILM agent is supposed to learn the knowledge of all these event detection algorithms in the feedback system and merge their knowledge into one single model that is able to perform the task of all those traditional event detection algorithms in the feedback system with a higher overall score. Two probabilistic (LLR voting) and one expert heuristic (sliding window distribution change) event detection algorithms are chosen to be embedded in the feedback system. Each of these algorithms is designed to use different electric signals as its input streams. The parameters of each, are adjusted in a way that the algorithm can identify their targeted appliances, i.e., different power levels. Figure 5.9 represents how these three algorithms are used in the feedback system.

Each of these embedded event detection algorithms is adjusted in its own specific way. Their input streams are different, they are modified to identify a specific group of appliances better

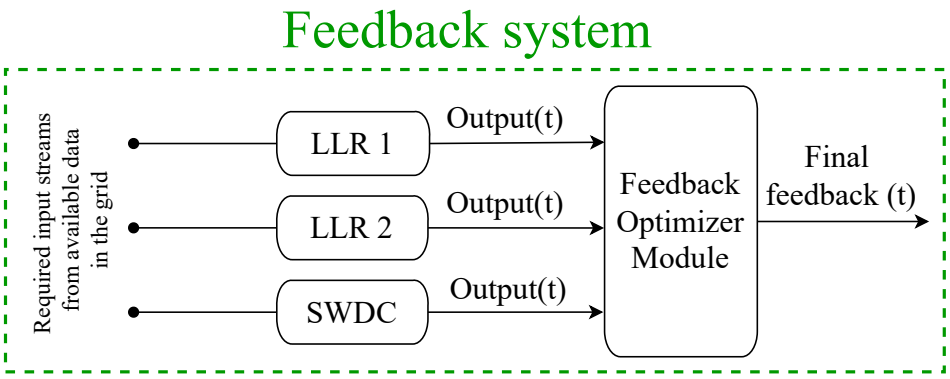

Figure 5.9 Event detection algorithms embedded in the feedback system

than the others. Thus, they make a great group of event detection algorithms to be tested in our proposed architecture of RLNILM. Table 5.1 presents the characteristics of each of these three test algorithms. For the ease of the readers, these three event detection algorithms are referred to as ED1, ED2, and ED3 for the rest of this document.

Table 5.1 The characteristics of three event detection algorithms used in the feedback system

Reference	Type	Parameters	Target	Initialization input
ED1	Probabilistic LLR voting	$w_0 = 5$ $w_1 = 5$ $\theta_P = 30$ $w_V = 10$ $\theta_V = 8$	Appliances with significant changes in their active power consumption, e.g., stove, iron	P at 1Hz
ED2	Probabilistic LLR voting	$w_0 = 3$ $w_1 = 3$ $\theta_P = 4$ $w_V = 6$ $\theta_V = 5$	Appliances with small changes in their reactive power consumption, e.g., water filter	Q at 1Hz
ED3	Expert Heuristic SWDC	$w_s = 10$ $\theta_{\delta_p} = 10$ $\theta_{\delta_q} = 5$ $\theta_u = 4$	Appliances with significant changes in both active and reactive consumption, e.g., fridge	P at 1Hz Q at 1Hz

As is shown in Figure 5.10, the feedback system and the RLNILM agent are being fed the same input streams from the environment (same states at every time step). However, each event detection algorithm in the feedback system reads the signal with which it has been trained. On the other hand, each event detection algorithm has its minimum or optimal operation input resolution (sampling frequency). Usually, if the provided input signals in the grid are

sampled at a higher frequency than the algorithm's minimum, there is no problem with the event detection algorithm's accuracy or performance score, However, when the frequency is lower than their set minimum, their performance score drops. In any case, the RLNILM agent is supposed to learn from the event detection algorithm in the feedback system and take the most out of them. In this experiment, different input frequencies are tried to investigate the importance of the signal sampling frequency on the learning ability of the RLNILM agent.

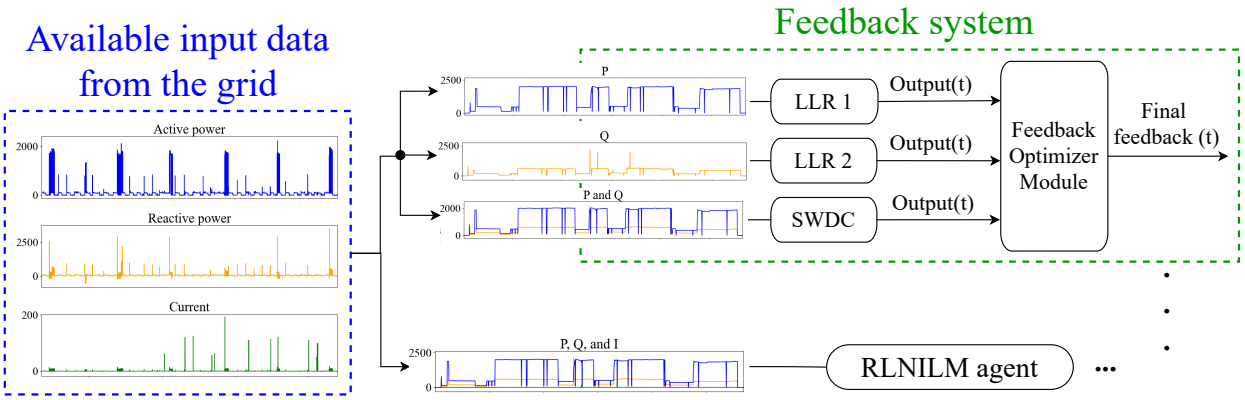

Figure 5.10 Environment states being fed to the feedback system and the RLNILM agent simultaneously

It is important to know the original performance score of each of these event detection algorithms (ED1, ED2, and ED3) before being implemented in the feedback system. In Table 5.1 the original sampling frequencies (and other parameters) of the chosen event detection algorithms are given. Each event detection algorithm is run on the same load profile (48 hours of the simulated iAWE dataset) and evaluated the performance of each algorithm which is shown in Table 5.2

Table 5.2 Performance metrics of embedded event detection algorithms in the feedback system

Accuracy of:	ED1	ED2	ED3
AC	93%	65%	69%
Fridge	95%	85%	82%
WF	10%	90%	46%
Iron	54%	68%	68%
$F_1 score$	78%	60%	64%

In the next sections, the metrics used for the evaluation purpose of the event detection algorithms are explained, however, as the accuracy and $F_1 score$ are used in Table 5.2, they are described as follows:

$$F_1 score = \frac{TP}{\frac{1}{2}(FP + FN) + TP} \tag{5.6}$$

where TP, FP, and FN stand for true positive, false positive, and false negative, respectively. $F_1 score$ is the most used metric in NILM event detection and not only represents how well an algorithm finds actual events but also it shows how well it can avoid situations that look like an event but they are not. A simpler metric, e.g., the accuracy is useful too in real-world cases in which we do not have access to appliance-level ground truth data.

$$Accuracy = \frac{TP}{N_{GT}} \tag{5.7}$$

where N_{GT} is the number of ground truth events of a specific appliance.

Even though it is not intended to get very deep into the scores of each of these algorithms here, we briefly explain the reason behind their different performances. It helps us better understand the value of unifying and optimizing the knowledge of all these algorithms by the RLNILM method. According to Table 5.1, ED1 and ED2 are two LLR-voting event detection algorithms. Even though their detection principle is the same, their sensitivity level is not. ED1 is designed to identify significant sharp changes in active power. ED1 avoids identifying small turn ON/OFF events due to its high power threshold (θ_P). Consequently, as it is observed in Table 5.2, ED1 is able to identify three appliance events with high active power changes but fails to identify water filter events due to its low power level. Thus, the accuracy is high for three appliances but it is very low for the water filter. However, the $F_1 score$ of ED1 is in an acceptable range because, despite so many missed events of the water filter, ED1 managed to identify most events of the other three appliances. For ED2, the situation is different due to its high sensitivity to power changes. It identifies too many events for all four appliances, so it is understandable that it correctly identifies most of the load events, however, the problem arises from its high sensitivity. ED2 identifies many non-event power changes as an event. Thus, numerous false positives are recorded. As a result of that, the $F_1 score$ falls. ED3 functions differently than ED1 and ED2. It reads two input streams P and Q. It scans these two streams to identify simultaneous changes in them. Because of this working principle, it is less sensitive to resistive loads, e.g., iron, and accurate for loads with a motor which creates sudden changes in both active and reactive power,r e.g., AC. This method is close to the event detection algorithm used by G. Hart in his first work on NILM [16]. This dual-stream check, helps the algorithm avoid false positive events while maintaining its sensitivity. Thus, its appliance detection accuracy boundaries are between ED1 and ED2 as well as its $F_1 score$.

Each event detection algorithm that is embedded in the feedback system generates a binary sequence where 1 is translated as an event (ON or OFF) at that time step and 0 means there is a no-event moment detected by the algorithm. Therefore, in this setting, there are three binary sequences as the output of these three event detection algorithms. The question is, how can it handle the possible contradictions in the outputs of the event detection algorithms at every time step? For instance, ED1 and ED2 may identify an event at a time step (i.e., their output at that time step is 1) and ED3 does not agree with them (outputs 0). How is it possible to unify and optimize their outputs in an expandable manner that can be applied to any number of event detection algorithms in the feedback system? To solve this problem the feedback optimizer module is proposed. The feedback optimizer module is supposed to read the output sequences of all embedded event detection algorithms in the feedback system and take a sequence out of that which is called the final feedback signal. This final feedback signal summarizes the knowledge of all used event detection algorithms and is used to train the RLNILM agent. It does not mean that the final feedback signal is always the same as the ground truth data, however, over the course of many repetitions in the training cycle of the RLNILM agent, it leads to a better training process than any of the original output sequences of event detection algorithms in the feedback system. Figure 5.11 is a flowchart that shows the logic of the feedback optimizer module.

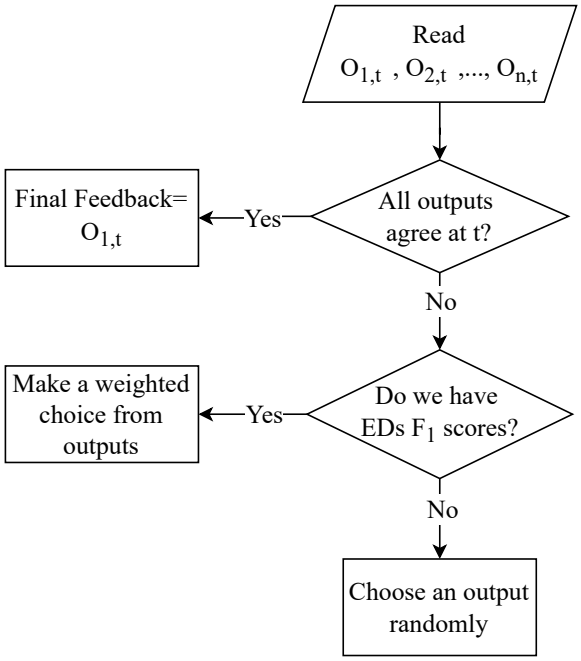

Figure 5.11 The working logic of the feedback optimizer module

According to Figure 5.11, the feedback optimizer module reads the output value of all n event

detection algorithms in the feedback system at time step t, i.e., $O_{1,t}$,$O_{2,t}$...$O_{n,t}\forall t$. Then, it compares the outputs. If all the output values are the same, i.e., all event detection algorithms detected an event at time step t, the final feedback signal becomes 1 which indicates the presence of an event at t, otherwise, in case of disagreement between event detection algorithms, the feedback optimizer module chooses the output of one of them based on a weighted distribution. It means that the event detection algorithm with a higher $F_1 score$ has a higher chance to be chosen, however, that is not guaranteed. There are other methods for this last step that is explained in the next chapter.

Figure 5.12 shows how the feedback module optimizer manages to generate a unique optimized signal from all three different output sequences that it gets from the event detection algorithms in the feedback system.

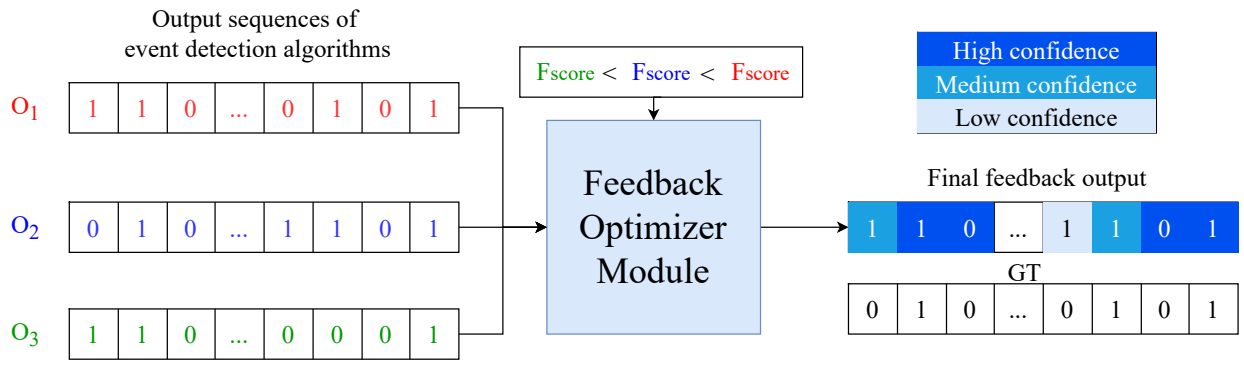

Figure 5.12 The performance of the feedback optimizer module in the presence of three event detection algorithms with different $F_1 scores$

Looking at Figure 5.12, it can be seen that at some time steps, the event detection algorithms have different opinions. GT stands for ground truth labels. Their event detection ability and sensitivity vary based on their working principles and their parameter adjustment. However, the feedback optimizer module checks the opinion of all of them and in case of disagreement, it makes a random but educated guess based on their performance score which is reflected in Table 5.2. Even in real-world cases, it is not difficult to run an event detection algorithm on a limited test data set with available ground truth data and calculate the evaluation metrics such as $F_1 score$. Thus, the performance scores of the event detection algorithms in the feedback system can be considered as some parameters for the RLNILM algorithm which should be obtained in advance.

Making an educated guess based on the $F_1 score$ of the event detection algorithms can be done in many ways. The simplest way to illustrate that is to correspond the $F_1 score$ percentage of each event detection algorithm to the number of repetitions of its output in the guess

vector. Thus, for ED1, ED2, and ED3 with $F_1 score$ of 78%, 60%, and 64% respectively, we can consider 5.8 as the guess vector from which we can choose a random value to have a weighted guess on the final feedback value.

$$\text{Guess vector}(t) = [\underbrace{O_{1,t}, \cdots, O_{1,t}}_{78}, \underbrace{O_{2,t}, \cdots, O_{2,t}}_{60}, \underbrace{O_{3,t}, \cdots, O_{3,t}}_{64}] \tag{5.8}$$

Before finishing this section, it is important to clarify why this part of the algorithm is called the feedback system. The reason behind this naming is the fact that the reward function generates the sequence of rewards for the RLNILM agent based on what it reads from the output of the system and the final feedback. The final feedback signal contains the knowledge of all event detection algorithms embedded in the feedback system and these algorithms get their input from the environment. Thus, in fact, the processed data that is injected into the RLNILM agent as a reward sequence stems from the outside of the feedback system. For this reason, naming it the feedback system is more meaningful than any other role or name.

In the next section, it is explained how the final feedback value is used in the reward function to transfer the knowledge of the event detection algorithms to the RLNILM agent.

5.3.5 Reward function

As the last part of the feedback system, it is necessary to discuss the reward function. The reward function has two inputs at every time step. The first input stream is the final feedback sequence from the feedback optimizer module and the other sequence comes from the output of the RLNILM algorithm (RLNILM agent actions). The reward function is responsible for judging the actions of the RLNILM agent by comparing them with the final feedback signals. In other words, the reward function is checking if the action (a_t) of the RLNILM agent in every state (s_t) matches the final feedback signal or not. In the previous section, it is explained how the feedback optimizer module boils down the outputs of the event detection algorithms into one optimized final feedback signal. The reward function employs the final feedback signal to create the reward signal (r_t) that teaches the RLNILM agent the knowledge contained in the event detection algorithms' output.

In general, reward functions in the RL algorithm may have many forms. They may be dynamic and complex functions that change based on the state of the problem, or they may be static and simple, e.g., being constant during all states of an episode. These two types of reward functions have no inherent superiority over each other. The algorithm designers should find out which method works best for their intended purpose. There are some points

to consider when designing a reward function for the RLNILM algorithm:

- The goal is to encourage the RLNILM agent to only identify true positive events
- The number of no-event time steps is much more than the number of event time steps
- Missing an event time step is much worse than confusing a no-event time step with an actual event time step

Considering the above-mentioned points, a simple table can be defined, e.g., Table 5.3, as our reward function. Keep in mind that these values are not mathematically calculated. They are estimated through a trial and error process while observing the accumulated reward of the RLNILM agent as in Figure 5.13.

Table 5.3 Rewarding strategy of the reward function

Agent action	Final feedback	Reward
Event (1)	Event (1)	250
Event (1)	No-Event (0)	-20
No-Event (0)	Event (1)	-250
No-Event (0)	No-Event (0)	10

It is true that the values of the accumulated reward curve in Figure 5.13 strongly depend on the reward function values in Table 5.3, however, its steepness and number of iterations to reach its maximum, helps us to find the best reward function for our RLNILM algorithm.

The reward sequence generated by the reward function helps the RLNILM agent learn what it should do to maximize its accumulated reward. In the next section, it is thoroughly explained how the RLNILM agent is formed and how it learns to optimize its actions when it faces different states and reward sequences from the environment.

5.4 RLNILM agent

The RL agent can be considered the most important element in any RL algorithm. Unlike the rest of the RL algorithm parts that function according to a constant instruction, the agent evolves and improves itself during iterations. Thus, in the first iterations which may be on the scale of thousands, the RL agent is just randomly interacting with the environment. After enough initial interactions, the agent begins to optimize its policy in order to maximize its accumulated reward. In this section, we go through the details of the structure of the proposed RLNILM agent.

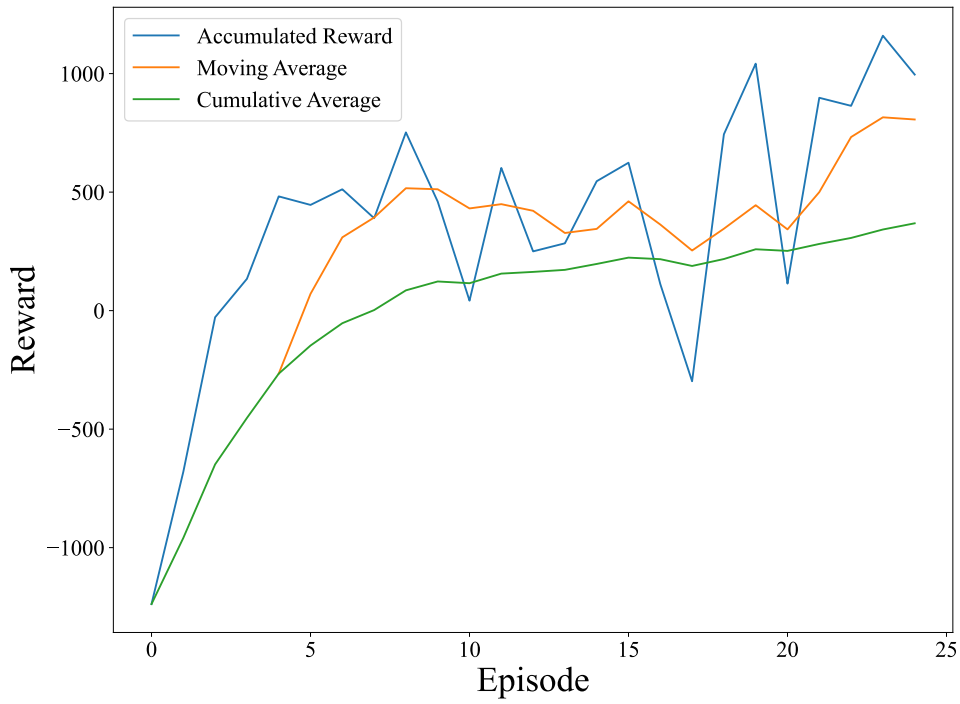

Figure 5.13 Accumulated reward of the RLNILM agent through several episodes during training

5.4.1 Input signals

In previous chapters and sections, different parts of an RLNILM algorithm are explained. Considering the agent as the most important part of an RL algorithm, it is important to know how the RLNILM agent is connected to other parts of the algorithm and what kind of data they exchange. The input of the RLNILM agent is a set of all available signals from the environment. In fact, both the feedback system and the RLNILM agent have access to all the signals from the environment, however, the event detection algorithms embedded in the feedback system, only read the input streams for which they have been designed to work with. For instance, ED1 which needs active power as its input has access to reactive power and current values at that moment too, however, they are useless to ED1 as it can generate an accurate output sequence by using active power values. In contrast, the RLNILM agent has no limit or preference to read a specific signal as its input. The RLNILM agent reads all available data streams from the environment and learns from them. Figure 5.14 illustrates this difference between the required inputs of the RLNILM agent and the feedback system.

It is worth mentioning that even though the frequency of the input streams for the feedback system and the RLNILM agent is the same (as they stem from the same environment), the

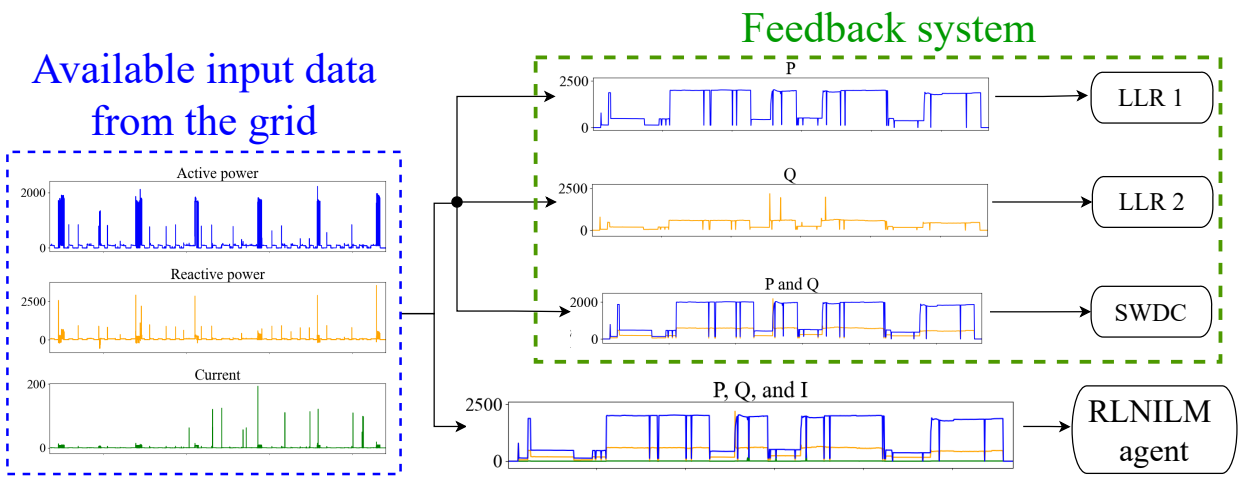

Figure 5.14 The input streams for the RLNILM agent and the feedback system

RLNILM agent has no preference for the sampling frequency, unlike the event detection algorithms in the feedback system.

Looking at Figure 5.10 gives us an opportunity to imagine how RL structure helps us to extract all the information from available input data signals. In this experiment, P,Q, and I are environment signals. However, imagine weather information is available at every time step, along with other electrical signals. How can weather information which has not been used by any of the event detection algorithms in the feedback system, be useful to the RLNILM agent decision-making? The answer is the correlation of non-electrical signals, e.g., weather or room occupancy information with the probability of usage of a specific appliance by the user. Traditional event detection algorithms are not able to understand and learn from these correlations, while the RL algorithm is capable of relating these changes in its input signals and learning from them. As a clear example, imagine a case in which the user turns ON the lamp in the bathroom and immediately turns ON the bathroom fan. A traditional event detection algorithm like ED1, identifies these two separate events if only they are separate enough, i.e., they happen with enough time distance. However, an RLNILM agent which uses the room occupancy data in addition to P, has learned that these two appliances are usually turned ON at the same time when the bathroom is occupied. Thus, because of this extra information, the RLNILM algorithm is probably capable of identifying these two events, even if they happen at the same moment. It was not possible to design such a test due to the lack of access to meaningful occupancy or weather information along with power consumption data, however, this concept can be very useful in appliances type classification in any NILM solution.

The above-mentioned electric signals are not the only input stream that the RLNILM agent receives. The reward sequence is another input stream that the RLNILM agent is fed with, at every time step. The electric signals are three windows of P, Q, and I values, with a certain window length, and the reward sequence is a vector of the last action's reward. Figure 5.15 represents all the input signals of the RLNILM agent.

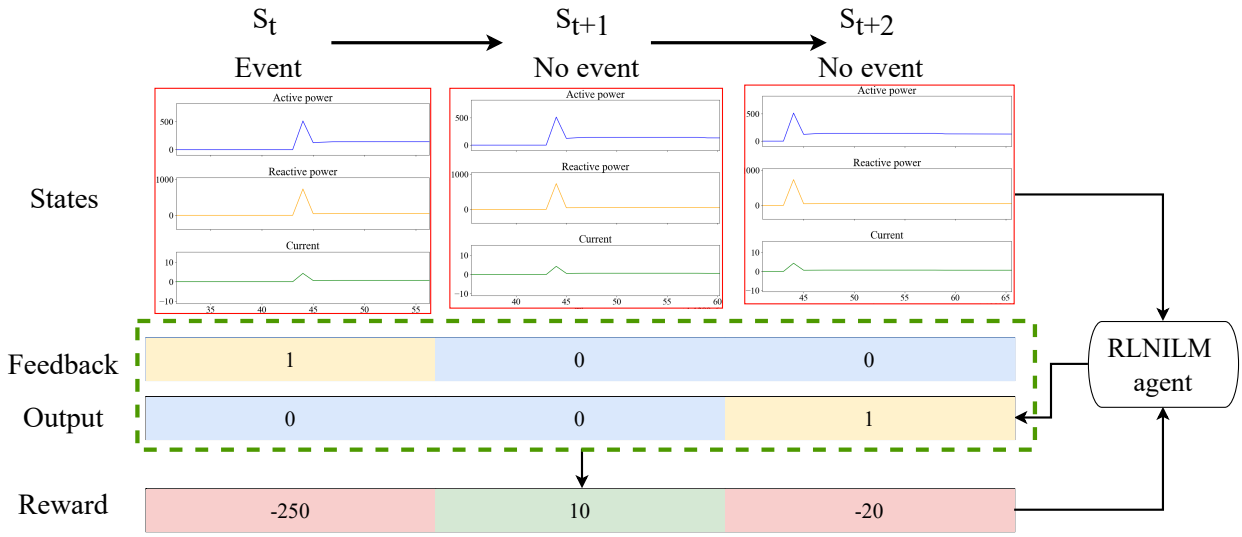

Figure 5.15 Electric signals and reward sequence as the input stream of the RLNILM agent

5.4.2 Preprocessing

Most ML algorithms need data preprocessing and RLNILM is no exception. Data preprocessing helps the algorithm run faster with a higher chance of convergence. In general, one may need to check data quality before feeding it to any algorithm. Missing data, errors, or wrong order, scale, and distribution are some of the problems that should be addressed in advance. The data from the iAWE dataset was cleaned and checked by the author of the dataset, thus, it is not needed to deal with missing data or any other abnormality. However, in this study, we applied three steps of preprocessing on our data for better performance of the algorithm.

The first step is to differentiate the values of all three signals (P, Q, I). These differentiated values are much easier for the RLNILM agent to understand as the main task of the RLNILM agent is to identify load events that strongly correlate with significant value changes in electric signals. The following equation and Figure 5.16 show how these values are generated:

$$\Delta X = X_t - X_{t-1} \tag{5.9}$$

where X represents any of the input signals P, Q, or I.

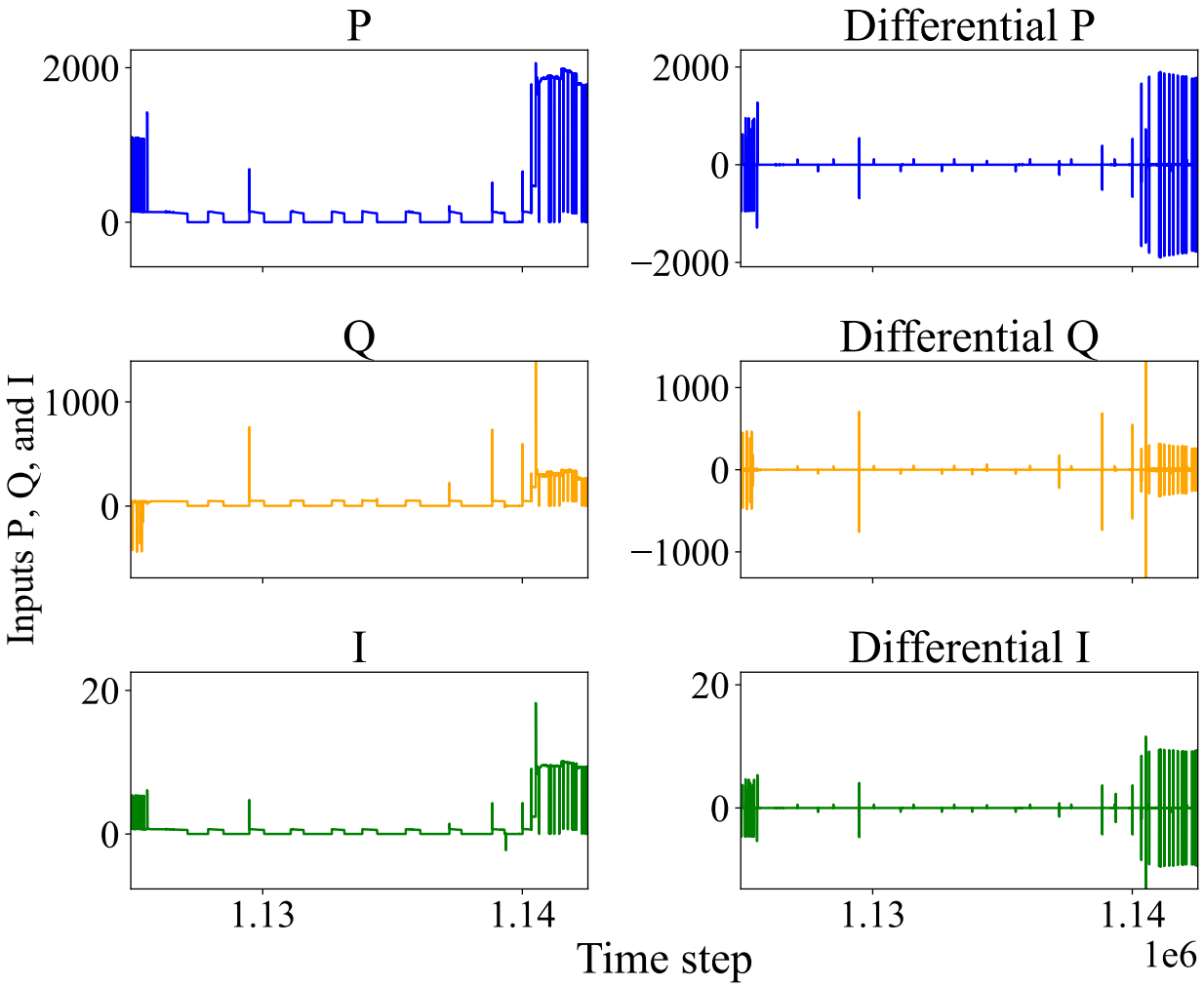

Figure 5.16 Differentiated inputs vs their actual values

We can observe that the differentiated signals are much easier to interpret for identifying the load events, even by looking at them. The next step in data preprocessing is standardization which means removing the mean and standard deviation of each window. Even though the differentiation step has removed the mean of the majority part of the data, this step helps to neutralize the deviation of data in each window of signals. The following equation standardizes the data.

$$X_{\text{standardized},t} = \frac{X_t - \mu}{\sigma} \tag{5.10}$$

where μ is the mean of the window of signal X, and σ is its deviation. The last step in data preprocessing for the proposed RLNILM algorithm is normalization. As it is shown in Figure

5.16, the scale of P, Q, and I are very different. The current is on the scale of a few amps and active power is on the scale of kilowatts. This diversity of ranges makes it very difficult for any ML model to learn anything from those signals. In order to overcome this issue, data normalization is done as in the following equation:

$$X_{\text{normalized},t} = \frac{X_t - \min(X_t)}{\max(X_t) - \min(X_t)} \tag{5.11}$$

That summarizes all the requirements for the input signals of the RLNILM agent. The decision-making process of the RLNILM agent can be discussed next.

5.4.3 Decision making

We briefly discussed in Chapter 4 how an RL agent can improve its learning through several repetitions of the state-action-reward-state cycle. In this section, the process is explained in details.

The goal of the agent is to maximize its cumulative reward. It means that the agent should take an action when encountering a state, that brings the highest total reward in the long run for the agent. The agent makes this series of decisions based on its logic. This logic or plan is called a policy. Unlike a map, a policy is not necessarily constant during the time. In fact, the improvement of the agent's decision-making process comes from the improvement of its policy. State-action value is the value of taking action a when in state s, for achieving the highest cumulative reward possible. State-action value of action a at state s is represented by $Q(s, a)$ and we call it Q-value [45]. This explanation can be summarized in an equation as follows.

$$Q(s, a) = r(s, a) + \gamma \sum_{s'} p[\pi(s)] V^{\pi}(s') \tag{5.12}$$

where $r(s, a)$ is the immediate reward of taking action a given the state s, $\gamma \in (0, 1)$ is the discount factor which determines the importance of future rewards for the agent (determines how short-sighted the agent is), p represents the probability of the transition from state s to s' under policy π, and V^{π} is the state value. Previously $V^{\pi}(s)$ was defined as the value of being in state s for the agent to maximize its total reward. Thus, having all the Q-values for every possible (s, a) combination, is in fact a comprehensive road map for the agent to maximize its cumulative reward.

In simple problems where an RL agent is supposed to find the best solution (best policy), we are dealing with a limited number of states and actions. Thus, all the situations in which

the agent will be can be illustrated with a table which is called Q-table. Table 5.4 illustrates such a table where all Q-values corresponding to different states and actions are determined.

State \ Action	a_1	a_2	$\cdots$	a_m
s_1	Q_{s_1,a_1}	Q_{s_1,a_2}	$\cdots$	Q_{s_1,a_m}
s_2	Q_{s_2,a_1}	Q_{s_2,a_2}	$\cdots$	Q_{s_2,a_m}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_n	Q_{s_n,a_1}	Q_{s_n,a_2}	$\cdots$	Q_{s_n,a_m}

Table 5.4 An illustration of the Q-table for a finite-state, finite-action problem

In Table 5.4 the first column on the left represents all the possible states. The top row shows all the possible actions in each state. In this example, it is assumed that all actions are feasible at all states, however, there is no obligation on the initial assumptions of the problem. For instance, in many problems, s_n represents the terminal state in which no more action is allowed because that state is in fact the end of an episode of the problem. Considering Table 5.4, Q_{s_i,a_j} shows the value of taking action a_j in state s_i. In our implementation, we can say that if the agent is in state s_i, then it will choose the action that has the highest Q-value. However, according to the exploration-exploitation strategy which is mentioned in section 4.3.2, sometimes the agent prefers to take a non-intuitive action to explore other actions instead of taking the most profitable action at that state. It means that sometimes the agent prefers to act differently from its policy and chooses an action that has a lower Q-value in order to explore new possibilities to improve the policy even more. Implementation of these settings of the algorithm is explained in the next sections.

If the Q-table is randomly initialized, and the agent starts the first episode from state s_j, it will choose the action corresponding to the highest Q-value in the first row of the table. The chosen action is denoted as a_i. The action a_i affects the environment. The environment sends back a new state $s_{j'}$ and a reward $r_{jj'}$ (the transition reward from state j to j') for the previous action a_i. Now, the agent knows that taking action a_i at state s_j has resulted in a reward of $r_{jj'}$ and puts the agent in state $s_{j'}$. Thus, the agent can recalculate all its assumptions written as its policy. Every time the agent takes an action, the resulting next state and the gained reward help it to recalculate the Q-value of that state-action pair which finally leads to policy improvement. The following equation shows how the new Q-value is calculated with every move.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t, a_t, s_{t+1}) + \gamma . \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5.13}$$

where t represents the time step, $\alpha \in (0, 1)$ denotes the learning rate of the algorithm and $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ is the maximum possible Q-value in future state s_{t+1} by taking future action a_{t+1}. According to 5.13, the updated Q-value $Q(s_t, a_t)$, is the sum of its non-updated value and a portion of the maximum possible reward of the future actions. Coefficient α adjusts the speed of Q-value changes toward this maximum (how fast the agent tries to learn Q-value changes). $r(s_t, a_t, s_{t+1})$ is the immediate reward for choosing action a_t and transition from state s_t to s_{t+1}. The middle part of the equation, $\gamma . \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$, adjust the greediness of the agent to achieve the maximum Q-value of the next state. Lastly, by subtracting the Q-value of the current state-action, the formula calculates the difference between the Q-values of time step t and $t+1$. This Temporal Difference (TD) way of updating the Q-table is one of the most common methods in RL [45].

Remember that the Q-table is not the actual reflection of the problem environment, but it is how the agent sees the problem. With every interaction between the agent and the environment, a Q-value gets updated and the entire Q-table gets closer to the actual problem environment. In fact, the agent is learning the actual Q-values of the environment, therefore this method is called Q-learning.

As it is mentioned before, in simple RL problems, the states and actions are finite and a Q-table can perfectly represent the perspective of the agent on the environment. However, in larger state-action spaces, e.g., our RLNILM algorithm, the situation is different. Usually, a larger state-action space means a continuous state-action space in which the number of states can be unlimited. In this situation, three different methods can replace the Q-table to represent the problem. The first method is using a simple n-dimensional grid to group input features by their ranges. Figure 5.17 illustrates this method. In this grid, any state that has features in a specific range will be represented by the corresponding tile in the grid. This method groups the continuous input features, thus, the number of states will be finite and more manageable. This method lowers the processing burden, however, it reduces the learning ability of the agents as the estimations are rough. A more accurate method has been introduced in the programming world and it is called tile coding. This method has similarities with the grid grouping method that is just introduced. However, this method is more accurate because it can represent many more states with fewer coordinates which means a lower processing burden. In tile coding, the entire state space (continuous state

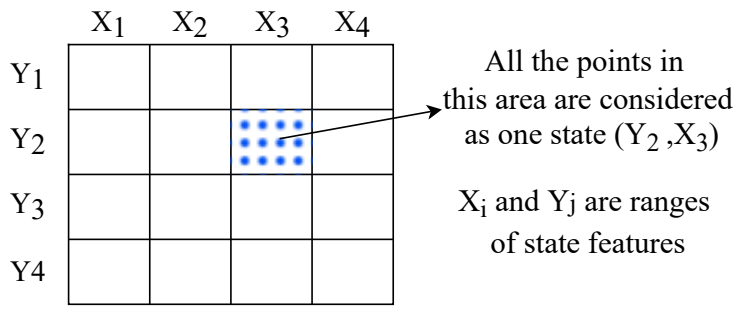

Figure 5.17 A simple 2-D grid grouping state estimation

space) is covered by overlapping tilings. Figure 5.18 shows a sample tile coding setting. In this example, four tilings are covering the majority of states with higher resolution than each of the tiling. In this setting, it is possible to point to any specific state group by indicating the coordinates of each corresponding tile in each tiling. As a result, we have much finer segmentation for the unlimited state space, with just four coordinates. The state estimation in code tiling is more accurate than a simple grid grouping method while it lowers the needed processing power for the algorithm. In code tiling, all those overlapping tilings are creating

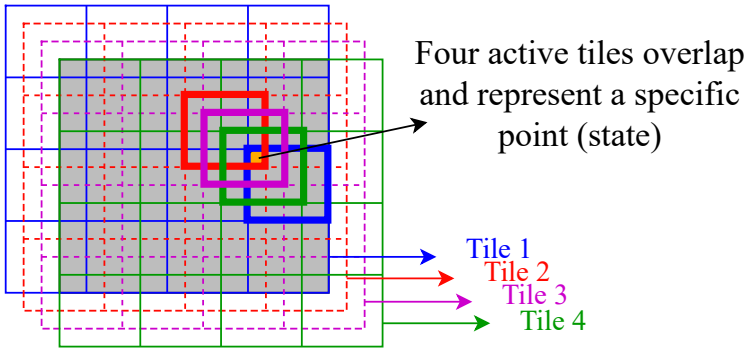

Figure 5.18 A tile coding sample for four tilings

a segmentation over the entire state space. Each segment contains many states which will be represented by the same tile coding. It is worth mentioning that the shape of tilings can be different which may create different net patterns (instead of evenly distributed squares all over state space) in the state space.

Despite all benefits of the code tiling method for segmentation of the continuous state spaces, there is still a better way to reduce the complexity of the Q-table in larger problems, however, the method itself gets more complex. This method is called function approximation which is implemented using Artificial Neural Networks (ANN). In fact, this is a replacement for the

policy or Q-table. Thus, instead of having a table in which all the states, possible actions, and corresponding Q-values are listed, there is a neural network that approximates the Q-values based on the state features (input layer) and takes an action by assigning a higher value (estimated Q-value) to one of its possible actions (output layer). The Q-values are calculated in the network (hidden layers). As this is one of the main parts of our RLNILM agent structure, let's get deeper into its structure [45].

Neural Networks are not new concepts in machine learning, however, they are significantly more used since the improvement of processors in recent years. We are not going to cover the working principles of neural networks as they have been discussed in many studies. In this part, it is explained how neural networks are used as a Q-table approximation in our proposed algorithm [45].

First, we create a neural network which is called the policy network. The policy network's role is to observe the input features (state), estimate the Q-values, choose the action based on the exploration-exploitation strategy (ϵ-greedy), and finally optimize its matrix of weights and biases (by doing backpropagation). Figure 5.19 illustrates the general structure of the policy network. The policy network has an input layer, some hidden layers, and an output layer.

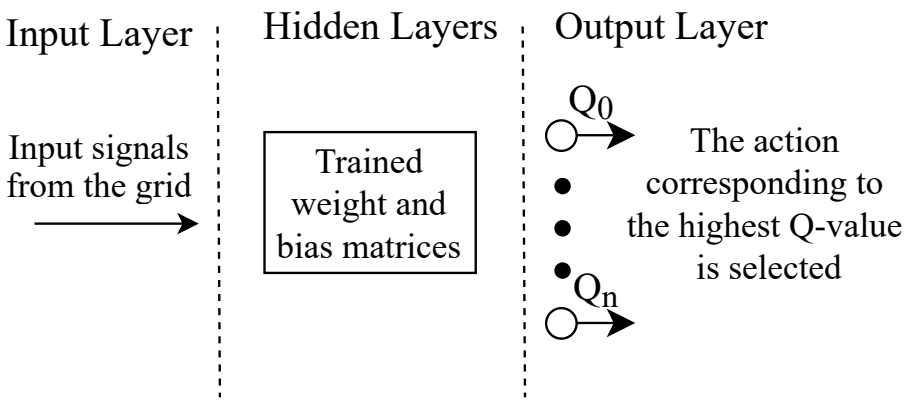

Figure 5.19 The general structure of the policy network

In the input layer, three vectors of P, Q, and I are fed as the input features (representing the state). Hidden layers and their weights and biases are supposed to use these input features to estimate the Q-values for the RLNILM agent, and also to choose an action (an output) that yields the highest cumulative reward in the long term. One of the questions in designing every neural network structure is the number of nodes and layers. One may imagine that the more complex the neural network, the better it works. However, it is not the case. Every problem has its own unique conditions for which a neural network should

be optimized to converge to a solution. As a convention in machine learning problems, to find the best-performing structure for a given problem, we trained the network with different combinations of parameters (number of layers n_l, number of nodes in each hidden layer $n_{h_{1,2,3}}$, and the learning rate lr). To keep it relatively fast and efficient, a set of numbers are defined to run the test using them instead of running over all the integers in a range which makes the optimization process very long. The following table shows the numbers that are used for the neural network structure optimization process:

Table 5.5 Parameters values used in neural network optimization process

Parameter	Variable	Value
Number of nodes	n_{h_i}	8,16,32
Number of hidden layers	n_l	2,3
Learning rate	lr	0.001,0.0001,0.00001

Several supervised models were trained using neural networks and these parameters. We chose the best-performing structure for the neural network structure. The best-performing parameter combination for our problem's training data was $n_l = 3$, $n_{h_1} = 8$, $n_{h_2} = 16$, $n_{h_3} = 16$, and $lr = 0.001$. The output layer which is in fact the layer of possible actions consists of two nodes. One node represents a "no-event" time step (and its value is the state-action value of a "no-event" action at that time step) and the other represents an "event" moment in the input state. The full technical settings of the used neural network are presented in Table 5.6.

Table 5.6 The state approximation neural network parameters

Parameter	Value
Number of hidden layers	3
Number of hidden layers nodes	8/16/16
Number of input nodes	3X20
Number of output nodes	2
Learning rate	0.001
Hidden layers activation	Relu
Output layer activation	Linear
Optimizer	ADAM
Batch size	32
Loss function	MSE
Cross-validation	30%

Thus, the optimized structure for the policy network can be shown in Figure 5.20. One may raise a question about the actual difference between the performance of all the possible

parameter combinations in Table 5.5. It is true that in one iteration the difference is not much, however, in RL which requires thousands of iterations, it is crucial to optimize the neural network as much as possible. During our test, before network optimization, we faced situations in which the network never converged or took so much time that made any more exploration impossible.

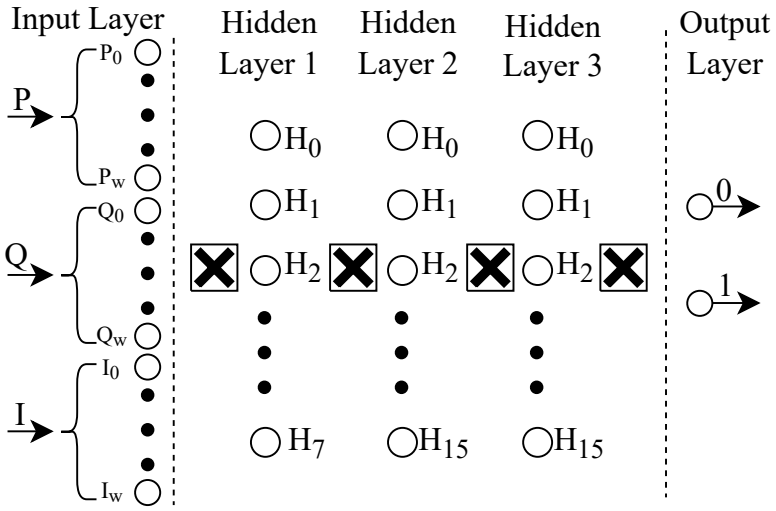

Figure 5.20 The structure of the optimized policy network

Knowing the structure of the policy network and the meaning of its input and output, it is easier to explain how it can learn from the feedback system that is introduced earlier in Section 5.3.4. The state features are sent from the environment to both the RLNILM agent and the feedback system. Event detection algorithms in the feedback system read their inputs and generate their outputs which show their verdict on detecting an event on that time step or not. Their verdicts then are unified (optimized) in the feedback optimizer module and the final feedback signal is generated and sent to the reward function. At the same time RLNILM agent has read the inputs from the environment (state) and the policy network generates values in its output nodes. The node with the higher value represents the chosen action by the RLNILM agent. The RLNILM's action and the final feedback signal are both sent to the reward function and based on its rewarding strategy, a reward value is sent to the RLNILM agent. Having that in mind, we can explain how the RLNILM agent uses this numeric reward value to improve its estimation of Q-values which means improving its knowledge about the event detection problem.

From 5.13, we remember how Q-values are updated in each iteration. The following equation represents the extended equation for updating Q-values recursively.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t, a_t, s_{t+1}) + \gamma . \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5.14}$$

Considering 5.14, it is noted that we have $Q(s_t, a_t)$ from the policy network's output layer, and $r(s_t, a_t, s_{t+1})$ from the reward function. The only variable that should be obtained is $Q(s_{t+1}, a_{t+1})$ which represents the Q-value of the next time step. The question is how can we estimate the next time step's Q-value and update the current time step's Q-value estimation using that (look at the temporal difference part of 5.14)? How can we optimize the policy network's weights and biases using 5.14 ? The solution is to fix (or save) the weights and biases of the policy network for a while until it calculates the next time steps' Q-value. Remember that the weights and biases matrix of the policy network is being updated (because of the backpropagation process) at every iteration.

To overcome this crucial problem another neural network is created and is called the Target Network. It is initialized with the matrix of weights and biases of the policy network. Thus, at first, they are the exact same network with different names. Having a parallel network keeps the algorithm much faster than the situation in which we save and read the weight and biases matrix every time. It is also mentioned that in 5.14 Q-values of the next time step are needed while we just have access to the current time step information. To have the next time step information it is possible to shift everything one time step back. Because it does not change the sequence of information, it does not affect the learning process of the RLNILM agent. Thus, when time step t is mentioned, it means the previous time stamp and when we mention $t + 1$ or the next time step, it is in fact the current time step. However, to keep it easy to understand, the naming convention is not changed.

The policy network is supposed to read the current state (s_t) as input and generate Q-values for each action in that state ($Q(s_t, a_t)$). The target network is supposed to read the next state (s_{t+1}) and generate the next time step state-action values ($Q(s_{t+1}, a_{t+1})$) from which we can choose the maximum value as the best possible action for the next time step. If both the policy network and the target network update their weights and biases in every iteration, 5.14 can not improve the Q-values estimation of each time step. It happens because the policy network is changing according to the Q-values generated by the target policy which is also changing. Thus, they are both changing at the same time and for that reason, the learning process can not converge to a real solution and it changes its direction at every iteration. In order to let the policy network truly improve its Q-values estimation, the target network estimation of the next time steps Q-values needs to stay constant for a while so the policy network can improve its Q-values estimation using the temporal difference term in

5.14 ($\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$). In fact, this is the reason why this parallel network is called the target network because it generates an estimation of the next time steps Q-values with which the policy network improves itself. To make this happen a parameter for iteration delays (d) between policy network updates and target network updates is considered. In our proposed algorithm we set $d = 400$, however, it can be different based on the nature of the problem and other parameters. It means after every d iteration of the policy network, the new weights and biases matrix of the policy network are copied to the target network. Figure 5.21 gives us a better understanding of this process

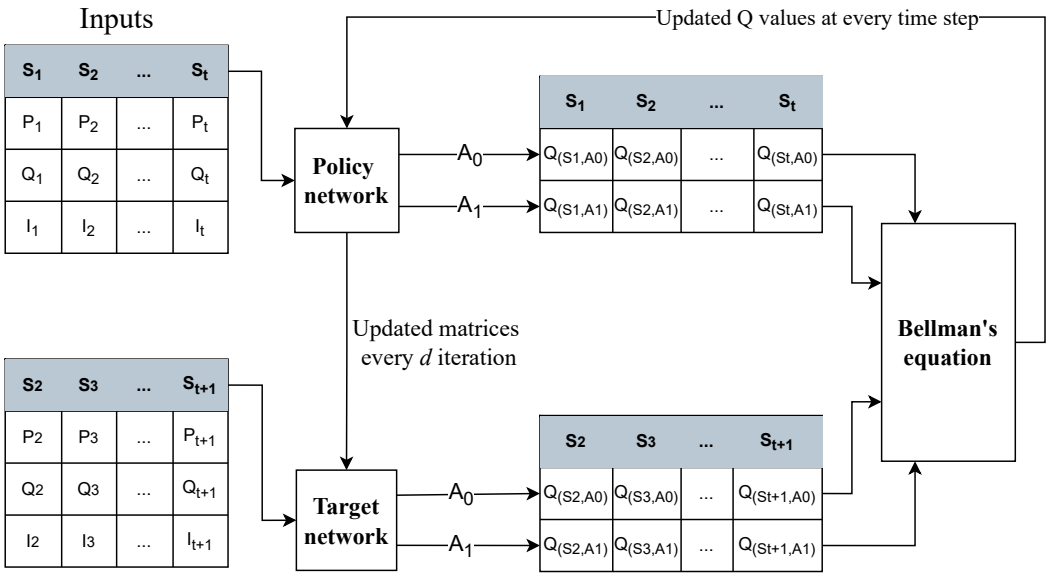

Figure 5.21 The process of updating target network with policy network weights and biases matrix

In Figure 5.21 it is observed that the policy network reads the input features (current state) at every time step and assigns a Q-value to each action in its output layer (event or no-event). At the same time step, the target network generates Q-values estimation for the next state. Policy network uses the target network Q-values to update its Q-values estimations using 5.14. The policy network uses the new Q-value for the backpropagation process and updates its weights and biases. Thus, the policy network is possibly improved a little bit compared with its previous version. However, the target network does not change at all. The target network never does the backpropagation process because its task is to generate the estimations of the next time step Q-values for the latest version of the policy network which is updating itself at every iteration. After enough iterations (in our case, d), the policy network has improved enough and it is converging to a solution. At this stage, we copy the weights and biases of the policy network into the target network and update it to the next

version of the policy network. With this weights and biases matrix update, we can be sure that the target policy is also able to provide more accurate Q-value estimations for the next time steps which will lead to more improvement and better learning for the policy network. This cooperation between policy and target network is what makes it possible to converge to a solution (learning when an event happens), otherwise, the policy network would get lost in its steps toward finding a solution to minimize its loss function as it is using a future value (next time step maximum Q-value) while all are changing in every iteration. Not fixing the target network is like following a moving target which is less likely to converge to a solution compared to a temporarily fixed target.

5.4.4 Exploitation vs. exploration

In the previous part, it is explained how the RLNILM agent is making decisions to identify events. It is also briefly mentioned that besides the agent's policy for choosing the best action at each time step, there is another strategy that helps the agent discover better actions at each state. This strategy is called the exploitation-exploration strategy [45]. It means that sometimes the agent chooses the action that leads to the highest Q-value possible (according to its Q-table), and sometimes it decides to choose a random action except for the action with the highest Q-value. Adding this small strategy to the main policy of the agent helps it to find the optimal solution faster with a lower risk of being trapped in local minima [45]. Imagine a case in which the random initialization of the agent's policy is in a way that the best action in a certain state has a low initial Q-value. As a result, that action either gets never selected in that episode or it takes a long time before the agent tries other actions with a higher Q-value and finds out that their actual value is lower than their initial Q-value, and then if the episode is not over yet, maybe the actual best action gets selected. Figure 5.22 illustrates how the real best action may not be discovered without the exploitation-exploration strategy.

To avoid such a situation, the exploitation-exploration strategy is defined. Exploitation simply means letting the agent choose the action which has the highest Q-value. On the other hand, exploration means that the agent chooses another action randomly to see what is its actual outcome. It is a common approach to set a ratio at which the agent chooses a random action. This ratio is high at first and as the agent interacts with the environment, in later iterations, it is reduced. At first, the agent has no knowledge about the environment, and all the Q-values are chosen randomly, thus choosing the highest assigned Q-value is no different than choosing an action randomly. However, after each iteration, the Q-values are corrected based on the reward and outcome of that action. Therefore it makes sense to

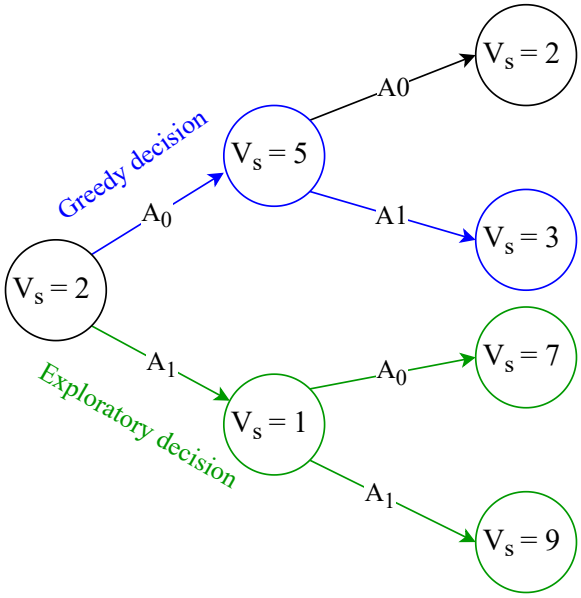

Figure 5.22 A situation in which the exploitation-exploration strategy helps the agent to find the best action

trust more in the agent's exploitation and lower the share of random actions. After enough iteration, it is safe to reduce the exploration ratio to a very low value because the agent has learned a lot about the environment and its Q-values properly represent the environment. However, we can always keep a small chance for choosing a random action with the hope of finding a better policy or hidden optimal solutions. Considering all of that, the exploitation-exploration strategy can be defined as 5.15 and it is called $\epsilon - greedy$ strategy [45].

$$\epsilon = \min \epsilon + [(\max \epsilon - \min \epsilon) \times \exp^{-\frac{\text{iteration}}{\text{decay}}}] \tag{5.15}$$

where min ϵ is the lowest probability of a random action being selected, max ϵ is the highest possible value for that, iteration represents the number of iterations up to that moment, and decay is a parameter to control the speed of exponential changes of the ϵ value. Figure 5.23 shows how ϵ is reduced when the number of iterations increases.

The RLNILM agent acts greedy (exploitation) on ϵ% of the iterations and explores other actions which do not correspond to the highest Q-value, for the rest of the iterations in a random order. Equation 5.16 represents this strategy.

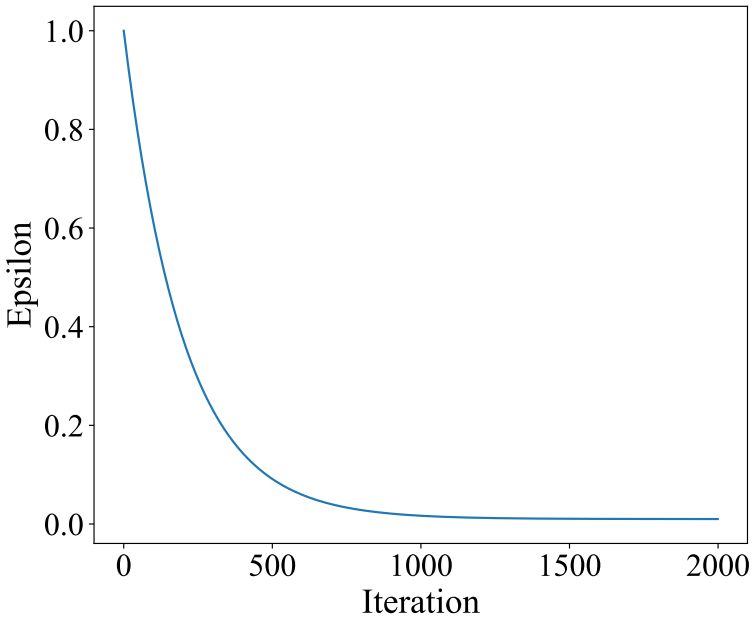

Figure 5.23 ϵ value decreases as the iterations go on

$$a_t = \left\{ \begin{array}{cc} \text{argmax}_{a_t} Q_{s_t,a_t} & \text{Uniform[0,1]} > \epsilon \\ \text{Unifrom[0 and 1]} & \text{else,} \end{array} \right\} \tag{5.16}$$

In the next part, it is shown how this complex process is actually happening. One of the most important parts of our proposed algorithm that plays the role of memory for a brain is introduced in the next section. The brain can not work properly without having access to an efficient memory that saves and categorizes the agent-environment interactions properly.

5.4.5 Dual Replay Memory

As it is discussed earlier, RL algorithms need a lot of iterations compared to many other machine learning algorithms [45]. Considering the fact that an RL agent acquires its knowledge by learning from interactions with the environment, the importance of having enough samples becomes bolder. Aside from having enough quantity of samples to learn from, the time that an RL agent needs to experience all those interactions to learn from, is also of great importance. Imagine a robotic arm that is controlled by an RL agent, is supposed to learn how to put small objects in a box. It is well-known that most RL algorithms need thousands of interactions to gain the required knowledge to solve a problem. Even if every episode takes just a few seconds, the overall time would be huge and makes the learning process difficult. On the other hand, there may be some limits for the maximum number of interactions of

the agent with the environment, e.g., a person is supposed to readjust the robotic arm after each test which makes the realistic number of possible experiments even lower than in theory. Thus, it is crucial to make the RL algorithms more sample efficient in order to learn the most out of a limited set of data samples.

To improve the performance of our RL algorithm a part called Replay Memory is added to it [45]. If the neural network is the decision-making part of a brain, the replay memory is its actual memory in which it saves the history of the most recent interactions with the environment. In the case of the proposed RLNILM agent, without a memory, the RLNILM agent has to train its policy network with one state per time step which is not efficient at all because it takes a very long time to learn something. However, by adding a memory the RLNILM agent is able to save the most recent interactions with the environment (state, action, next state, reward) and use them in batches many times to learn from. Doing forward and backpropagation with batches of the most recent data samples makes the process of learning much more efficient and faster, however, it becomes more compute-intensive.

The replay memory makes it possible for the policy network to learn from agent-environment interactions. It is called replay memory because it literally replays the previous data samples for the RLNILM agent. Figure 5.24 illustrates the simplest form of replay memory for the RLNILM agent.

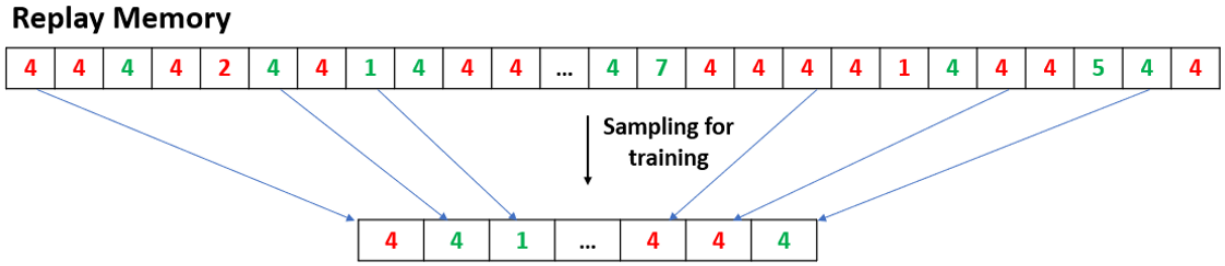

Figure 5.24 The simplest form of a replay memory for the RLNILM agent (green: correct prediction, red: wrong prediction)

According to Figure 5.24, the replay memory is an vector with a finite length l (in our case $l = 5000$) that saves a list like $[s_t,a_t,s_{t+1},r_t]$ in its cells. These l lists are the most recent interactions of the agent with the environment. One may ask how the parameter l is determined. The answer, again, goes back to the nature of the problem of interest or better to say, the problem for which we are designing an RL agent. A very long replay memory contains very old and inaccurate interaction data that may make it harder for the RLNILM agent to find the optimal solution. On the contrary, a short replay memory does

not contain enough fresh memories to help the RLNILM agent efficiently learn anything about the environment. This number is chosen by trial and error, considering the fact that there are many more "no-event" samples compared to "event" samples. For that reason, a long enough replay memory is needed to capture enough event samples.

At first, the policy network is initialized with a totally random weights and biases matrix. Then this matrix is copied to the target network. The policy network, as the decision-making part of the RLNILM agent, starts to interact with the environment states one by one. It reads the current state (s_t), generates an estimation about Q-values of its possible actions ("event" vs."no-event"), and based on the exploration-exploitation strategy it chooses either the action with the maximum Q-value (exploitation) or chooses the other action (exploration). The a_t is sent back to the environment. The feedback system compares a_t with its final feedback signal from its embedded event detection algorithms and rewards the RLNILM agent accordingly. Lastly, the environment sends the next state to the RLNILM agent. With that being done, the list $[s_t,a_t,s_{t+1},r_t]$ is complete and it is recorded as the first entry in the replay memory vector. This list is called an observation. This cycle goes on and on until the number of observations reaches the batch size. Batch size is the number of observations that the policy network takes at once to be trained with. Training a neural network with batches of data keeps the training process faster and increases the chance of convergence. According to Table 5.6, the batch size in the proposed algorithm is 32. The RLNILM agent keeps interacting with the environment until it records at least 32 (batch size) observations in the replay memory with the length of $l = 5000$. The replay memory length and the batch size are not unique and we determined them by trial and error. Now, the policy network reads a random batch of 32 observations from the replay memory to train itself. The policy network's weights and biases get updated and it is ready to read the next input features (the next state) from the environment to create another observation. The new observation is recorded in replay memory, and the cycle goes on. During this process, the policy network is becoming more accurate and it is learning more accurate Q-values of the problem which will lead to better performance of the RLNILM agent. Meanwhile, the target network stays constant and helps the policy network to update its Q-values. However, when the policy network updates itself $d = 400$ times (an intentional delay for the convergence of the policy network) the target network replaces its weights and biases matrix with those of the policy network. This cycle continues and finally, all $l = 5000$ cells of the replay memory will be filled with the agent's observations. At this stage, when a new observation comes in, the oldest observation in the replay memory pops out and the new observation fits in the replay memory vector. Figure 5.25 clarifies how these steps happen in sequence.

Thus, at every iteration one new state is presented to the RLNILM agent, however, as the

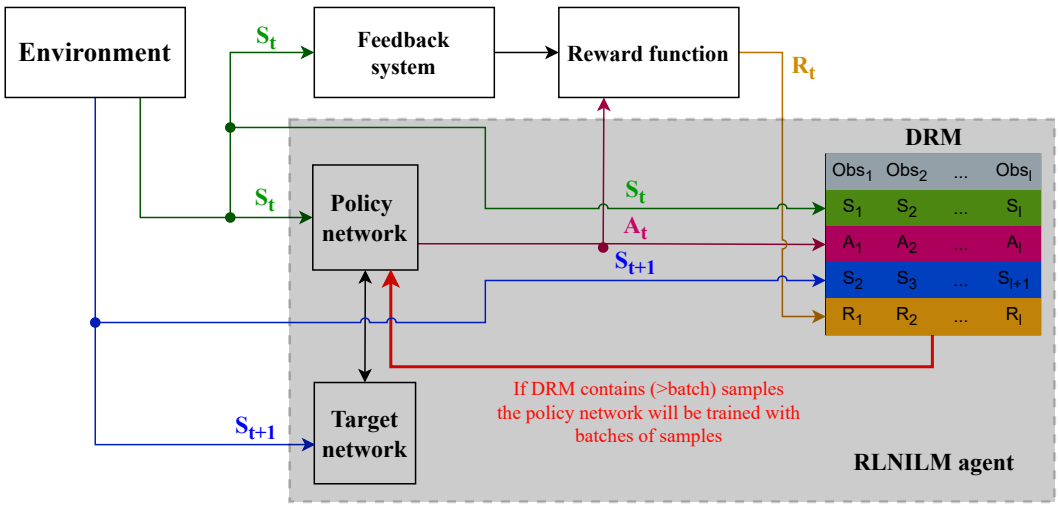

Figure 5.25 The interactions between the DRM and the rest of the system

RLNILM agent is reading a batch of 32 samples, it will train itself with 32 samples at every iteration instead of just one sample which makes the learning process more accurate (avoiding biases and divergence). On the other hand, each observation is possibly used more than once, thus the algorithm becomes much more sample efficient which means it can even converge to a solution with a lower number of samples. In fact, neural networks learn a little from each sample, therefore, it is necessary to present each sample a few times to the model. In order to take the most out of a sample, we change the combination of samples in batches too.

This architecture of replay memory works well in many RL problems, however, during our test, it is noted that this architecture of replay memory is not efficient enough for the event detection problem due to its nature. It is observed that the RLNILM agent is learning how to identify events in the first few thousand iterations which means its cumulative reward was increasing as the RLNILM agent iterates over the observations in replay memory. However, after a while, the learning curve (cumulative reward vs iterations) was falling and the RLNILM policy network was diverging from finding a solution. In RL literature, this issue is called "Catastrophic Forgetting". It happens when the model (RLNILM agent) is trained with too many samples that correspond to the same output, i.e., the situation in which there are too many of one state compared to other possible states. An example of an RL agent that is supposed to learn how to conduct a self-driving car, is helpful here. Imagine the test road that is used to train the RL agent is mostly a straight path and there are a few curves. At the beginning of the road, the agent learns that sometimes it should drive straight and sometimes it should turn right or left, thus its learning curve (cumulative reward) is increasing. However, as the path is mostly straight, after a while the number of "go straight"

observations becomes much more than "turn right/left" observations. Consequently, the RL agent learns that by driving straight it will gain a much higher cumulative reward even if it gets punishment (negative reward) for those moments that it was supposed to turn and it did not. This case (catastrophic forgetting) happens because the RL agent forgets what a mistake looks like, i.e., it is experiencing so many samples of certain cases that it forgets about what it has learned about more complex situations, and because those complex situations are rare, then the RL agent does not try to learn how to handle those situations.

It is important to understand how this issue happens in event detection problems. Figure 5.26 reminds us what a residential load profile sampled at 1 Hz frequency looks like. Remember that technically, an event happens at one time step which is usually surrounded by a sequence of no-event time steps. Thus, considering all the no-event time steps in a day, the ratio of the number of events over the number of no-event time steps would be a small number. The same problem as the self-driving car example happens here. In the first batches of training data, the RLNILM finds a balanced set of event and no-event cases. As the RLNILM agent interacts with the environment and more observations are recorded in the replay memory, the share of no-event cases increases. Now the RLNILM agent realizes that by choosing the no-event at all iterations, it can achieve a much higher cumulative reward than trying to identify events from no-events which may lead to more mistakes. For this reason, the RLNILM agent forgets how to identify events and it generates the constant output of "no-event". Let us recall Table 5.3. For the ease of the reader, we provided this table here again. Table 5.7 shows how the reward function is rewarding the RLNILM agent's actions. As you observe in this table, the reward for the cases in which the final feedback is 0 (no-event) is much lower than in 1 (event) cases. It basically shows that the event moments which are rare are more important than no-event moments that occur more frequently. This method of rewarding helps the RLNILM agent understand the importance of event moments, however, the occurrence of no-event moments is more, to the extent that this method can not prevent the RLNILM agent from falling into the trap of catastrophic forgetting. To efficiently overcome this problem we introduced a new architecture for the replay memory which solved this issue for the RLNILM agent.

Table 5.7 Rewarding strategy of the reward function

Agent action	Final feedback	Reward
Event (1)	Event (1)	250
Event (1)	No-Event (0)	-20
No-Event (0)	Event (1)	-250
No-Event (0)	No-Event (0)	10

In our proposed RLNILM algorithm a new replay memory is designed by the author that is

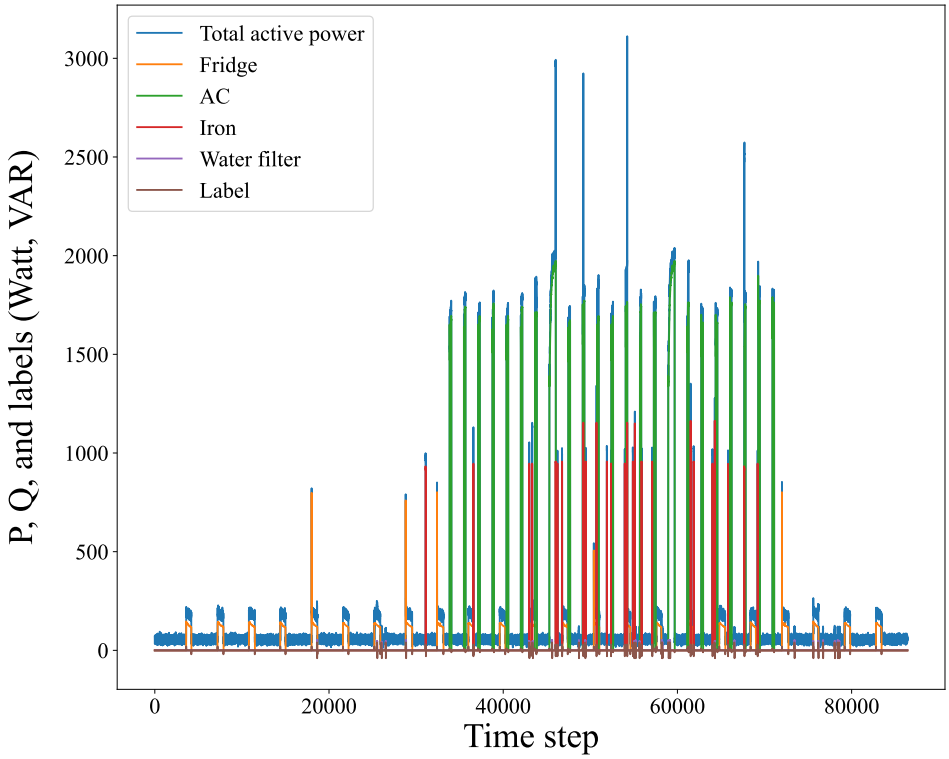

Figure 5.26 A load profile of residential unit during a day

called Dual Replay Memory (DRM). DRM is designed to solve the catastrophic forgetting issue for cases with unbalanced states occurrence that have different importance. Figure 5.27 illustrates how DRM works.

Same as regular replay memory, all the interactions of the RLNILM agent and the environment (observations) are saved in a vector. To keep the illustration simple, some items of the observation list ,e.g., state, next state, and reward are ignored. We just mentioned the actions of the agent (0,1) in the cells and their color coding shows that if the action matches the final feedback signal (green) or the action contradicts it (red). Next, regardless of the correctness of the actions in each observation, the events ($a_t = 1$) are saved in the event memory vector and no-events ($a_t = 0$) are saved in no-event memory vector. In the next step, the observations from both memories are mixed at the rate α as in 5.17.

$$\alpha = \frac{\text{Number of events}}{\text{Number of no-events}} \tag{5.17}$$

We chose $\alpha = 3$, i.e., for every no-event observation, we added three event observations to the

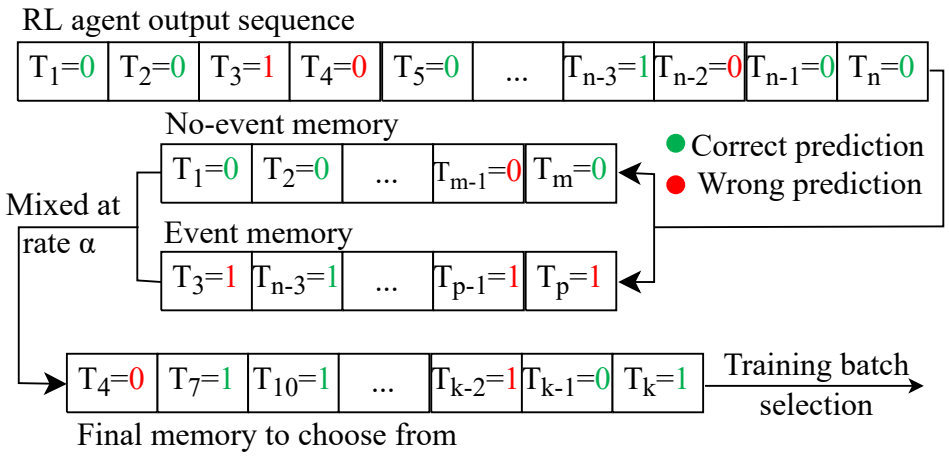

Figure 5.27 Dual Replay Memory workflow

combination. Thus, with every iteration, such a combination is added to the final memory from which the training batches are selected for the RLNILM agent. The reason why we chose 3 events for each no-event observation is that the event observation is in different forms for different appliances while most no-event states are like a flat noisy line. Therefore, the RLNILM agent should have more chances to be trained with various event states rather than no-event ones. This architecture for the RLNILM agent's memory provides the RLNILM agent with a faster and more efficient learning process which keeps it away from catastrophic forgetting.

5.5 RLNILM algorithm full cycle

In previous sections, different parts of the RLNILM algorithm are discussed in detail and explained how each part of the algorithm plays a role in the bigger picture. To conclude this chapter this is important to see how all these parts fit side by side to make the full cycle of the RLNILM algorithm work. Algorithm 1 summarizes this process and the sequence of different modules working together.

It is seen that there are several parts in the RLNILM agent's structure, working together to create a neural network model that learns the knowledge of other event detection functions embedded in the feedback system. The RLNILM agent saves the best model based on its performance (the highest $F_1 score$), thus, we can use the latest model to detect events in a load profile. When a new load profile is presented to the RLNILM agent by the environment, it adapts itself and starts to learn from this new load profile and after enough iterations, it is able to perform the event detection task as it is expected.

Algorithm 1: DRM RLNILM algorithm

Result: Identifying load events
Initialization and parameters;
Initializing the policy network with random weights and biases;
Copying policy network weights and biases matrix into the target network;
Initializing replay memory vector with agent-environment interactions (S_t, A_t, R_t, S_{t+1});
Agent-environment interaction;
for *each episode* **do**
    Initializing the environment;
    for *each timestep* **do**
        Agent takes an action according to the exploration-exploitation strategy;
        Environment reacts to the selected action by generating S_{t+1} and R_t;
        if *(A_t,R_t) represents No-event* **then**
            store (S_t, A_t, R_t, S_{t+1}) in No-event memory vector
        else
            store (S_t, A_t, R_t, S_{t+1}) in Events memory vector
        end
        if *both arrays can provide a batch* **then**
            Sampling a random batch of both memory vectors at the mix ratio of α;
            Passing S_t of the batch to the policy network as the input;
            Passing S_{t+1} of the batch to the target network as the input;
        end
        Q-learning process starts;
        The policy network chooses A_t considering calculated Q-values (output) and exploitation-exploration strategy;
        The reward function compares the feedback signal and A_t, and calculates the R_t;
        Calculating the difference between the chosen Q_{s_t,a_t} and the maximum $Q_{s_{t+1}}$ of the target network;
        Calculating new Q_{s_t,a_t};
        Making a new observation with the updated Q-value and corresponding values and pushing it to the replay memory ;
        Choosing a new batch from the memory replay to train the network policy;
        Minimizing the loss function of the policy network by ADAM optimizer and updating policy network weights;
    end
    After d episodes, update the target network's weights and biases to the policy network weights and biases;
    Saving the event detection model;
    if *F_1score improves* **then**
        Save the policy network as a model
    else
        pass
    end
end

CHAPTER 6 EXPERIMENTAL AND SIMULATION RESULTS

In the previous chapters, we talked about many aspects of the event detection problem in the NILM industry and how the proposed RLNILM algorithm improves the event detection process. We explained the principles of RL and how the proposed RLINLM agent solution is based on that. In this chapter, we go through the results of our experiment and analyze them to see how the proposed RLNILM solution improved the event detection process compared to the embedded event detection algorithms in its feedback system.

We applied the proposed RLNILM solution on the real-world iAWE dataset. The iAWE data set is explained in Chapter 2 and is used as the reference data for the performance evaluation of the event detection algorithms in this experiment. We tested the performance of the RLNILM solution on the iAWE data in four different scenario groups.

We have organized this chapter in the following order. First, we explain the evaluation metrics that are used in our experiment. Second, we explain the parameters and factors whose change creates different scenarios for the experiment. Then, we investigate the results of applying the RLNILM algorithm to different scenarios with the iAWE dataset.

6.1 Evaluation metrics

The goal of an event detection algorithm is to identify the time steps in which an event happens. Thus, a good evaluation metric for this purpose is F_1score which shows how the event detection algorithm is performing. The following equation shows how F_1score is calculated.

$$F_1score = \frac{TP}{\frac{1}{2}(FP + FN) + TP} \tag{6.1}$$

where TP, FP, and FN stand for true positive, false positive, and false negative, respectively. Our artificially-generated data from the iAWE dataset contains the event indicator vector (key vector) that shows the type of the transition (ON or OFF) and the appliance as in Table 2.1. Despite that, in event detection, we are not trying to find out which appliance is in which transition. The only thing we want from an event detection algorithm is to tell us the time steps of the load events as accurately as possible. However, we use the key vector to see which load transitions have been identified most often by our proposed RLNILM algorithm.

To calculate F_1score we need to know the number of occurrences of TP, FP, and FN. It

is not easy to define an exact moment of turning ON or OFF for appliances as the sampling frequency impacts what we see in a load profile. However, a few time steps ahead or behind does not affect the performance of a NILM system. For this reason and to keep the event detection concept practical, we define a threshold θ_t. θ_t is the number of time steps around an actual event occurrence time step in the load profile that we consider the event detection algorithm's output as acceptable event detection. In our evaluation process, we consider $\theta_t = 3$ which means if an event is detected three time steps before or ahead of a ground truth (actual) event, then that detection is considered TP. In this method of evaluation, the higher sampling frequency raises the expectation from the event detection function to perform more accurately than in a case where the algorithm is exposed to a low-resolution load profile. FP happens when the algorithm indicates to have detected an event while that time step is at least θ_t time steps away from any ground truth event. Finally, FN is the case in which a ground truth event is neglected by the algorithm, and nothing is reported. Figure 6.1 illustrates all of these three cases.

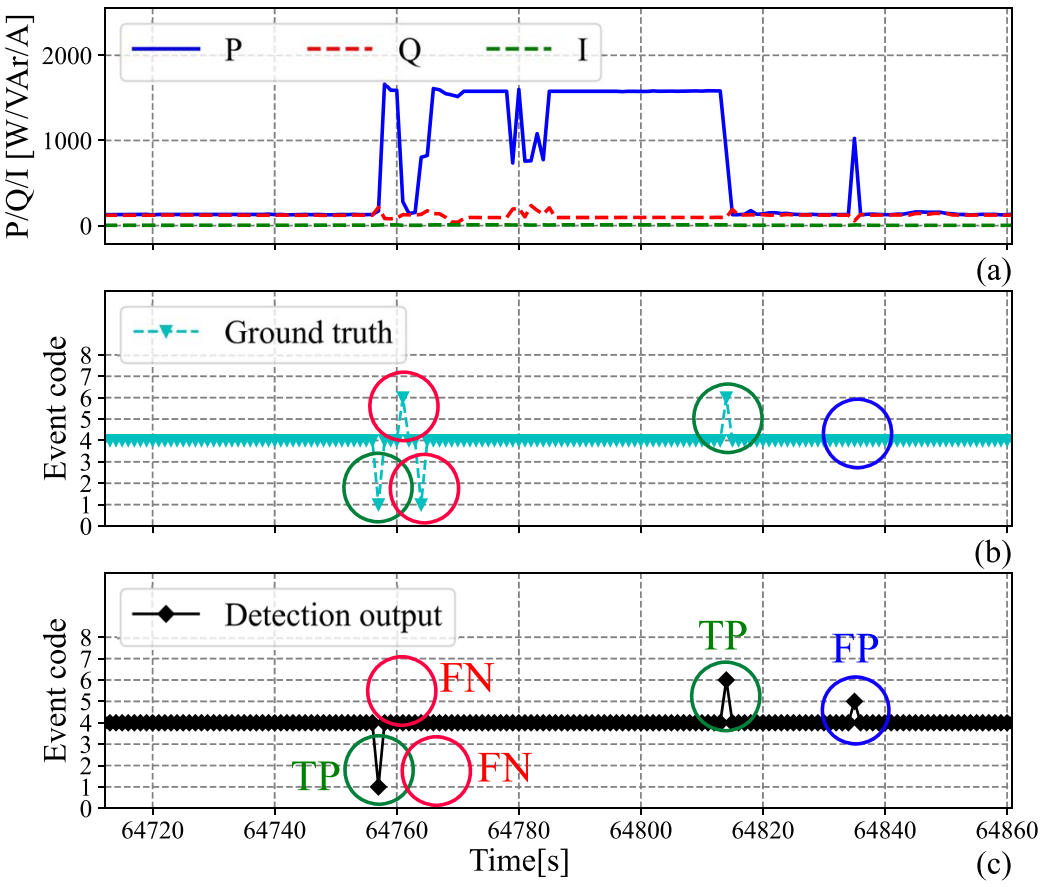

Figure 6.1 True positive, false positive, and false negative cases in event detection process

The next metric for evaluating the performance of the RLNILM algorithm in different scenarios is the appliance detection rate (accuracy). With this metric, we want to know how accurately various appliance signatures are being detected by the RLNILM algorithm in different scenarios as their signatures have significant differences in some cases. The formula for appliance detection rate is simple. The following equation represents this rate.

$$\text{Appliance detection rate} = \frac{TP_{A_i}}{N_{A_i}} \tag{6.2}$$

where TP_{A_i} is the number of correctly detected events of the appliance A_i and N_{A_i} is the total number of events of appliance A_i. This metric is calculated for all appliances in every scenario.

6.2 Scenario main parameters

In this section, the parameters that create different scenarios are explored. We have four main groups of scenarios that are called $S1$, $S2$, $S3$, and $S4$. Each scenario group contains seven sub-scenarios. The reason that they are called sub-scenario is the fact that they have a smaller impact on the results of each scenario. This categorization keeps the investigation of scenarios more organized for the reader. The next section explains the parameters that create these scenarios and sub-scenarios.

6.2.1 Input signal type

In Chapter 3, it is discussed that each event detection algorithm may be designed to work with a specific electric signal(s). Therein, we made examples of three different event detection algorithms that work with P, Q, and (P, Q) as their input signals, i.e., they operate optimally when they are fed with those signals as their input streams. In Chapter 5, it is explained that RLNILM can work independently and regardless of its input streams, i.e. it is not limited to only a specific electric signal as its input in order to operate properly. However, its performance is definitely affected by the type of electric input signals. With that being said, we need to study how the RLNILM algorithm performs, facing a different set of electric signals as its input streams. This test shows the flexibility of the RLNILM compared to the traditional event detection algorithms that are solely designed to work with a specific input signal.

It is assumed that the traditional event detection algorithms in the feedback system receive their required input signals. However, to test the RLNILM flexibility in different situations,

we feed it with different combinations of input signals in each sub-scenario. The following set shows different signal combinations of electric signals in our dataset that are used as the input for the RLNILM agent.

$$\text{RLNILM agent input} = \left\{ \begin{array}{cccc} P & Q & I & \text{single stream} \\ PQ & QI & PI & \text{double stream} \\ & PQI & & \text{triple stream} \end{array} \right\} \tag{6.3}$$

Thus, event detection algorithms in the feedback system always receive their needed input signals, however, the input streams of the RLNILM agent differ in every sub-scenario.

6.2.2 Input signal sampling frequency

Aside from the type of electric signal, input streams have another important attribute which is the sampling frequency. Sampling frequency can affect the event detection algorithm performance significantly as the event detection algorithms are modified based on their input stream sampling frequency. If the input resolution is decreased, most event detection algorithms miss many small or sharp events, however, they can still detect some other events. To investigate the level of sensitivity of the RLNILM algorithm to the input signals sampling frequency, we test the algorithm with two different sampling frequencies. The normal test condition is 1 Hz which is the actual sampling frequency of the dataset. The dataset is down-sampled 5 times (0.2 Hz) which makes it harder for all the event detection algorithms in the test to properly identify load events. Despite the previous parameter (the type of input signals) which was only changed for the RLNILM agent, this parameter (change of the sampling frequency) should be applied to the input of all the event detection algorithms to have a meaningful output. It is not possible to compare two algorithms that one of them creates an output every second and the other one only outputs every 5 seconds. Figure 6.2 illustrates how input signals look like with a 1 Hz-sampling frequency versus 0.2 Hz.

The sampling frequency difference in sub-scenarios helps us to understand how frequency-sensitive each event detection algorithm is. On the other hand, it shows how much decrease in their performance happens by the change of the frequency.

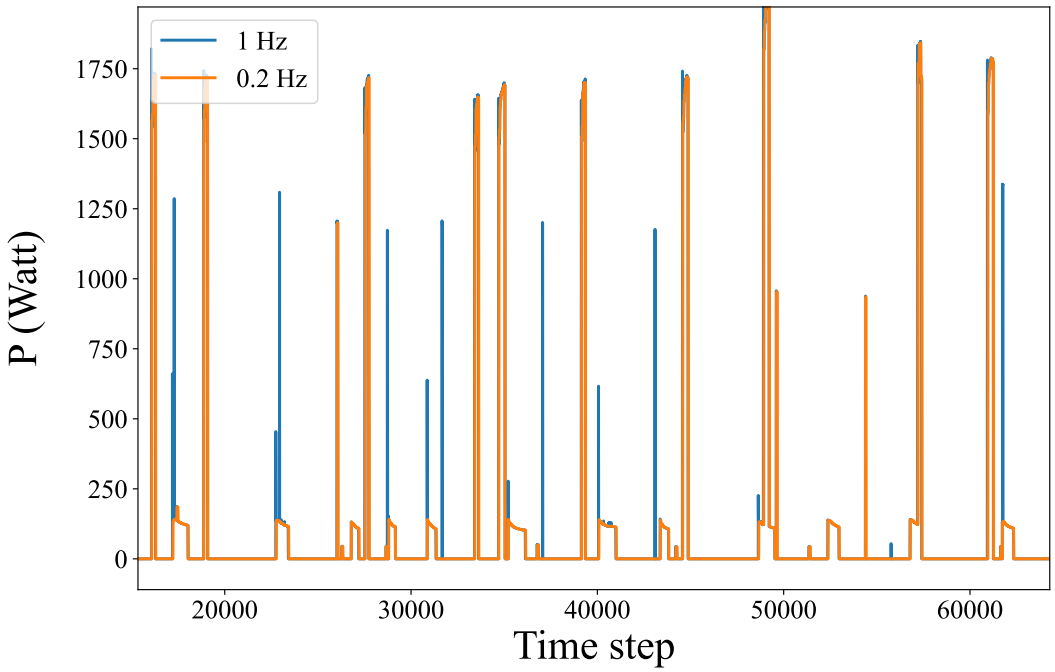

Figure 6.2 The same input signal, sampled at 1 Hz versus 0.2 Hz

6.2.3 Event detection confidence level

Earlier in Section 5.3.4 it is discussed how the feedback optimizer module, unifies and optimizes all the outputs of different event detection algorithms in the feedback system, and the reward function uses that signal to train the RLNILM agent. In that section, we assumed that the feedback optimizer module chooses the final feedback signal based on a weighted guess between the output of all the event detection algorithms in the feedback system. The weights of each output are in fact the $F_1 score$ of each event detection algorithm which is calculated separately and before integrating the event detection algorithm with the RLNILM structure. It means that the output of a better event detection algorithm (higher $F_1 score$) has more chance to be chosen as the final feedback signal. It makes sense, however, we want to challenge the performance of the RLNILM agent even more. An assumption can be that there has not been any opportunity to evaluate the performance of any of the event detection algorithms. Thus, we consider a case in which all the event detection algorithms are assumed to have equal performance. In this case, the chance of each of them to be chosen as the final feedback signal would be $\frac{1}{3}$. The chance of being chosen is called the confidence level. Thus, based on the confidence level of the event detection algorithms we have two sets of sub-scenarios, one set with confidence levels equal to event detection algorithms $F_1 score$ and the other set with equal confidence levels ($\frac{1}{3}$). This parameter helps us to understand how

the prior knowledge of the embedded event detection algorithms in the feedback system, can affect the RLNILM agent performance.

6.3 Defining scenarios

Considering the parameters mentioned in the previous section, we have the following sub-scenarios for the dataset as in Table 6.1.

Table 6.1 The parameters of all sub-scenarios

Scenario No.	Input signal	Sampling frequency	Confidence level
1	P	1 Hz	$F_1 score$
2	Q	1 Hz	$F_1 score$
3	I	1 Hz	$F_1 score$
4	PQ	1 Hz	$F_1 score$
5	QI	1 Hz	$F_1 score$
6	PI	1 Hz	$F_1 score$
7	PQI	1 Hz	$F_1 score$
8	P	0.2 Hz	$F_1 score$
9	Q	0.2 Hz	$F_1 score$
10	I	0.2 Hz	$F_1 score$
11	PQ	0.2 Hz	$F_1 score$
12	QI	0.2 Hz	$F_1 score$
13	PI	0.2 Hz	$F_1 score$
14	PQI	0.2 Hz	$F_1 score$
15	P	1 Hz	Equal
16	Q	1 Hz	Equal
17	I	1 Hz	Equal
18	PQ	1 Hz	Equal
19	QI	1 Hz	Equal
20	PI	1 Hz	Equal
21	PQI	1 Hz	Equal
22	P	0.2 Hz	Equal
23	Q	0.2 Hz	Equal
24	I	0.2 Hz	Equal
25	PQ	0.2 Hz	Equal
26	QI	0.2 Hz	Equal
27	PI	0.2 Hz	Equal
28	PQI	0.2 Hz	Equal

The combination of all sub-scenario parameters results in 28 sub-scenario. As the number of sub-scenarios is relatively high, comparing them in a meaningful way is of great importance.

In some plots, all the scenarios are compared, in some other plots we compare them in groups based on their shared attributes, and in some other cases, the best-performing cases of each group are compared.

6.4 Running the test and results

Before getting into the details of the sub-scenarios, it is helpful to take a look at the comprehensive plot of the RLNILM algorithm performance when it is fed by PQI signals sampled at 1 Hz frequency and the confidence level of $F_1 scores$. It is helpful to generally see how all different parts of the RLNILM algorithm are performing at the same time and how this cooperation leads to better results. Figure 6.3 shows all input signals in plot (a). Then, we have the output vector (decision vector) of all three event detection algorithms in plot (b). The final feedback signal which is the optimized version of output vectors in the upper plot, is shown in plot (c). Finally, in plot (d), we see how the RLNILM agent is detecting events versus the ground truth data of load events.

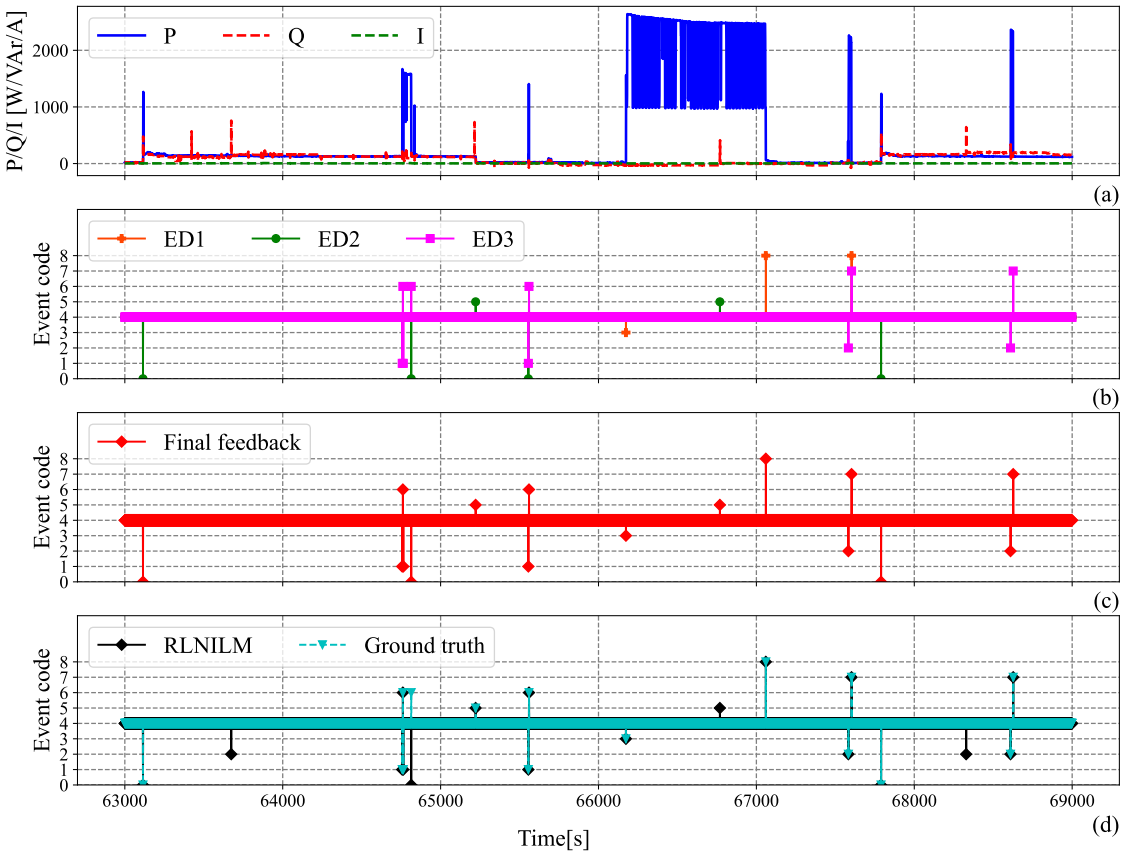

Figure 6.3 The plot of performance of all parts of the RLNILM algorithm at the same time

Having a clear image of how all these parts of the algorithm are working together and what are the inputs and outputs, we can discuss the result of all the sub-scenarios on the iAWE dataset. This experiment is run on a Lenovo T590 laptop with 16 Gigabytes of RAM and Intel core-i7 CPU @ 1.9 GHz. The algorithms are programmed with Python 3.7 Tensorflow library. To get the results of each sub-scenario, we set the sub-scenario's parameters in the problem and set the environment to generate 40 episodes with which the RLNILM agent is trained. After the RLNILM is trained, the saved model is used to be evaluated as the final model. The model is uploaded on a neural network with the same structure and 24 hours of input data is fed to it as a test. As the ground truth data is available, we can calculate all the necessary evaluation metrics which are discussed earlier in this chapter. Keep in mind that the main set of parameters to run the test are as follows:

- Input signals: P and Q
- Sampling frequency: 1 Hz
- Confidence level: $F_1 score$

By "main parameters", we mean that these parameters were chosen because of the original condition of the dataset and the design of event detection algorithms. The event detection algorithms that are used in the feedback system are optimized with P and Q streams as their input signals. As the iAWE data set sampling frequency is 1 Hz, the event detection algorithms are optimized at this frequency too. On the other hand, as we have access to the ground truth data of the input streams (in our test), $F_1 score$ of different event detection algorithms are calculated. Thus, the confidence level of different event detection algorithms in the feedback system is set to their $F_1 scores$. Therefore, this is our main sub-scenario and we pay special attention to its performance metrics in this chapter, however, to test the flexibility and vulnerability of the RLNILM algorithm to the change of these parameters, other tests are conducted with other sub-scenarios. First, the sub-scenarios are grouped as in Table 6.2, so it becomes easier to analyze them in plots.

Table 6.2 The parameters of all sub-scenarios

Group code	Sampling frequency	Confidence level
S1	1 Hz	$F_1 score$
S2	1 Hz	equal
S3	0.2 Hz	$F_1 score$
S4	0.2 Hz	equal

According to Table 6.2, code S1 represents all the sub-scenarios in which the sampling frequency is 1 Hz and the confidence level of all event detection algorithms in the feedback system is considered equal. Obviously, our main sub-scenario fits in group S1 where the sampling frequency is 1 Hz and the confidence levels are set to the $F_1 scores$.

6.4.1 Scenario group: S1

We start with the scenario group that contains the main sub-scenario. In this sub-scenarios, the input stream is sampled at 1 Hz frequency and the confidence levels are set to the $F_1 score$ of the event detection algorithm in the feedback system. The RLNILM agent is trained with different combinations of P, Q, and I as input streams. Figure 6.4 shows how different input streams affect the performance ($F_1 score$) of the RLNILM agent.

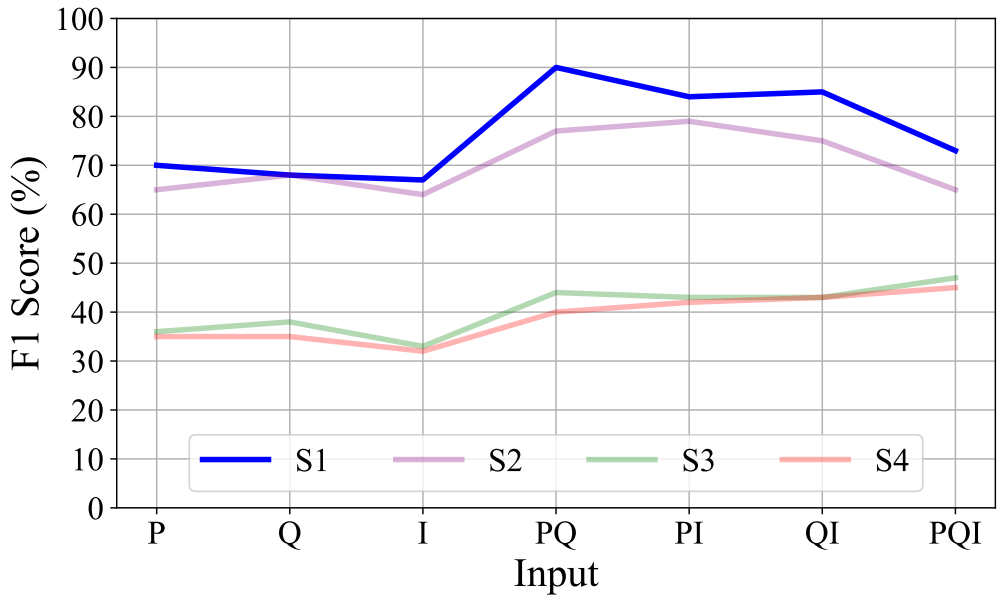

Figure 6.4 The $F_1 score$ of the RLNILM agent for different input streams in scenario group S1

Figure 6.4 indicates that the RLNILM agent works best with the PQ stream as its input, however, the other dual streams, e.g., PI, and QI have $F_1 scores$ a little lower than the PQ stream. It was expected, as the event detection algorithms are using both P and Q to generate feedback signals and by replacing one of them with I in the RLNILM agent's input stream, it is losing an important portion of data. Single streams, e.g., P, Q, and I are not performing well compared with dual streams for the same reason. They are not providing the RLNILM agent with the information that the event detection algorithm in the feedback

system has for decision-making. Between single stream inputs, I has the worst result and that was predictable because the feedback system is not using I in its event detection process. Lastly, we expected the PQI input stream to have the best performance, however, it turned out to be somewhere between dual and single inputs. Despite having the most information in this input stream, it seems that the RLNILM agent got overwhelmed with the amount of information that it is getting as input, and its performance dropped. It seems that having so much information with high frequency is confusing the RLNILM agent [46].

It is also important to compare the performance of the RLNILM agent in this scenario with the event detection algorithms in the feedback system to see if the RLNILM agent is able to improve their performance or not. Figure 6.5 illustrates this comparison.

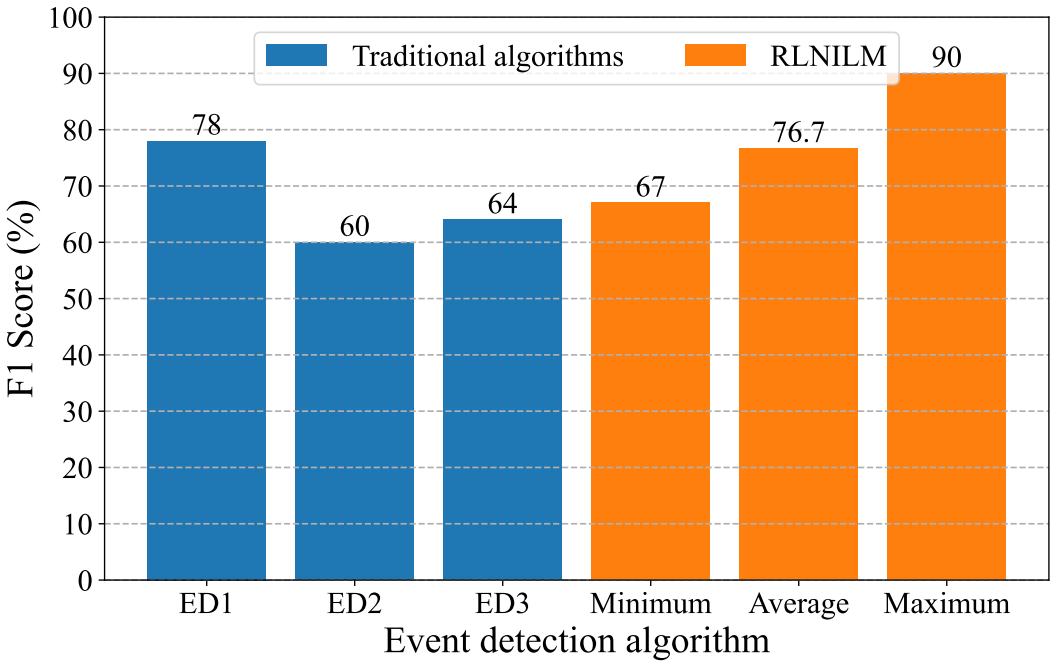

Figure 6.5 The minimum, average, and maximum $F_1scores$ of the RLNILM agent compared with event detection algorithms in the feedback system for scenario S1

Keep in mind that the event detection algorithms' performance is evaluated by their exclusive input streams (P, Q, PQ) and the change of input streams for the RLNILM agent training does not affect them. In this sub-scenario group, they are all receiving their input streams at 1 Hz frequency. According to Figure 6.5 the RLNILM agent outperforms all three event detection algorithms significantly ($F_1score = 90$) with PQ input stream, however, with a bad choice of input stream it falls behind all of them ($F_1score = 67$). On average the RLNILM agent has achieved the F_1score of 77 in this group of scenarios, taking into account all different input streams.

Having compared the performance of the RLNILM agent with the embedded event detection algorithms in the feedback system, it is insightful to see how the RLNILM agent is performing event detection on each appliance. Remember that the RLNILM agent only identifies if an event happens at a specific time step or not, and it can not tell anything about the type of the event (appliance). However, as the ground truth data of all events is available, i.e., we know what the type of every event is (AC ON, Fridge OFF, etc.), we can calculate the accuracy of the RLNILM agent's performance in detecting every event type. Table 6.3 shows the overall statistics of the test data which is fed to the RLNILM agent.

Table 6.3 Overall statistics of the test data

Variable	Quantity
Fridge ON	48
Fridge OFF	48
AC ON	23
AC OFF	23
Iron ON	11
Iron OFF	11
Water filter ON	32
Water filter OFF	32
Total events	228
Total no-event	86172

If the RLNILM agent has detected a turn-ON or turn-OFF of an appliance within the allowed threshold, we consider that detection a TP, and if it has missed the event, it is considered an FN. Remember it is not possible to define a FP for appliance-level evaluation because the output of the RLNILM agent only determines the occurrence of an event and not its type. Thus, for this reason, it is not possible to calculate $F_1 score$ for the appliance-level performance, however, the accuracy in event detection gives us a good insight into the efficiency of the RLNILM when facing different appliances. Table 6.4 shows the performance detail of the RLNILM agent on each appliance in this group of scenarios.

Table 6.4 The RLNILM agent performance on appliance event detection in S1 scnearios

	Fridge		AC		Iron		Water filter	
Input signal	TP	FN	TP	FN	TP	FN	TP	FN
P	82	14	43	3	16	6	51	13
Q	79	17	38	8	17	5	50	14
I	75	21	37	9	15	7	45	19
PQ	92	4	44	2	20	2	61	3
PI	86	10	45	1	19	3	58	6
QI	85	11	44	2	18	4	52	12
PQI	80	16	40	6	18	4	54	10

Herein, calculating the accuracy means the ratio of the TP which are correct guesses over the total number of events of the appliance which is TP+FN. Thus, by calculating the accuracy as in 6.4 we can conclude the Table 6.5.

$$\text{Accuracy} = \frac{TP}{TP + FN} \tag{6.4}$$

Table 6.5 The RLNILM agent accuracy on appliance event detection in S1 scnearios

Input signal	Appliance detection accuracy			
	Fridge	AC	Iron	Water filter
P	0.85	0.93	0.72	0.73
Q	0.82	0.82	0.77	0.79
I	0.78	0.80	0.68	0.68
PQ	0.95	0.95	0.90	0.96
PI	0.89	0.97	0.86	0.87
QI	0.88	0.95	0.81	0.84
PQI	0.83	0.86	0.81	0.84

From Table 6.5 it is observed that the PQ stream is the best-performing in the S1 group for all appliances. Figure 6.6 shows all the appliances' detection accuracy in this group side by side. It is obvious that increasing the number of input streams has increased the accuracy of the RLNILM agent for all the appliances. The iron and the water filter have a lower accuracy in all sub-scenarios in this group. It may be due to the shape of their signature which is much shorter (number of timesteps) compared with the AC and fridge which have longer load signatures. When a load signature is short, chances are that the load signature gets mixed with a noisy or transient part of the load profile and as a result, the algorithm can not identify that. On the other hand, short load signatures have turn-ON and turn-OFF moments very close to each other which in some cases may lead to the elimination of one

of the identified events by the event detection algorithm (remember that in some algorithms two events close two each other are merged). Lastly, the AC has the highest detection rate in all sub-scenarios of this group. It may be due to the amplitude of the rising and falling edge of turn ON or OFF events. Those edges are big enough not to be mixed with any other noise or not to be missed by any of these algorithms.

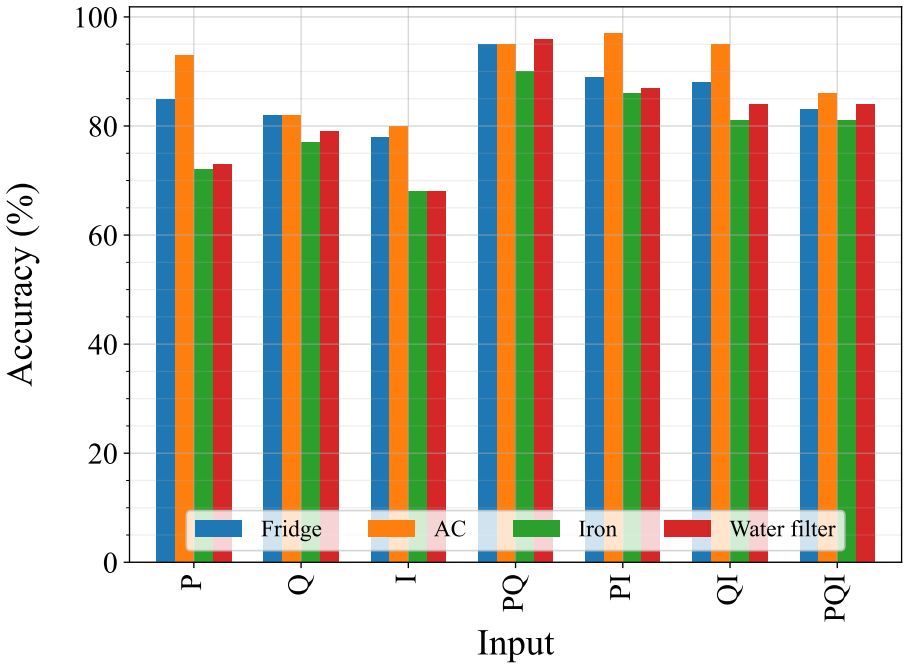

Figure 6.6 The RLNILM agent accuracy of detecting events of different appliances in scenario group S1

The performance of the RLNILM agent in identifying events is discussed more after we analyzed the results of all other sub-scenario groups. This group (S1) is actually the main scenario group. The rest of the groups are to test the flexibility and tolerance of the RLNILM agent in more complex situations.

6.4.2 Scenario group: S2

In this group of scenarios, the sampling frequency is 1 Hz, as in the previous sub-scenarios. However, in the feedback optimizer module, we change the confidence level of each event detection algorithm. In the previous group of scenarios, the weight of each event detection algorithm for the feedback system output generation was the same as its $F_1 score$. In this scenario group, the confidence level of all event detection algorithms is set to $\frac{1}{3}$, i.e., equal

values. The reason for this change is the fact that in many real-world power consumption datasets ground truth data of the events is not available. Even though it is possible to create that ground truth data, it is still time-consuming and in many cases complex. For taking these situations into account and testing the RLNILM agent performance without this important piece of information we run the test with this group of scenarios. Figure 6.7 shows the $F_1 score$ of the RLNILM agent with different combinations of input streams when applied to scenarios of group S2.

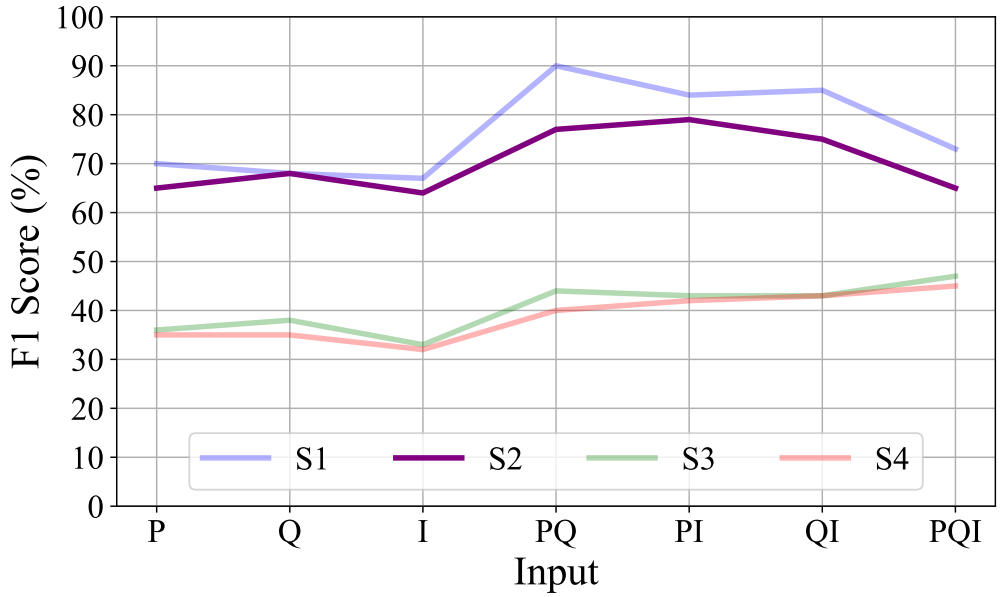

Figure 6.7 The $F_1 score$ of the RLNILM agent for different input streams in scenario group S2

As it is seen in Figure 6.7, single stream input scores are around the same value, even compared with the previous group in which the confidence levels were different. It shows that the single stream inputs contain less data compared with dual streams to the extent that even having different confidence levels is not helping the RLNILM agent to learn better. Dual stream inputs have higher scores and among them, PI has the best results, however, the difference between PI, PQ, and QI is not significant. It seems that losing the information of confidence levels of the event detection algorithm caused this decrease in $F_1 scores$. Lastly, it is observed that the PQI input stream has a low score almost as low as single streams. Overall, the absence of true confidence level values caused a problem specifically for multi-stream cases. When the confidence levels of all event detection algorithms are equal, it is like ignoring the performance of event detection algorithms in different situations and trusting

their output randomly. For this reason, the number of wrong decisions increases in this case and the learning process of the RLNILM is impacted. Figure 6.8 shows the performance of the RLNILM agent against all three event detection algorithms in the feedback system. As it is observed, in the best case, the RLNILM agent achieved a F_1score slightly better than the best-performing event detection algorithm in the feedback system. It shows the importance of the confidence level. In fact, the confidence level helps the RLNILM agent to learn from the most reliable feedback and not randomly-chosen ones.

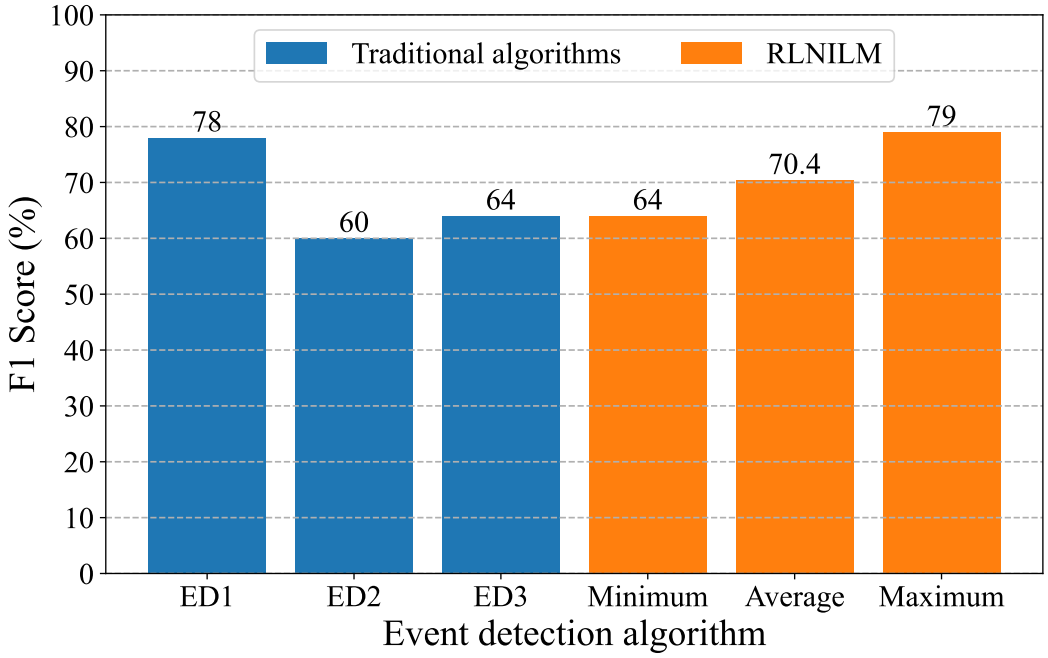

Figure 6.8 The minimum, average, and maximum F_1score of the RLNILM agent compared with event detection algorithms in the feedback system for scenario group S2

Table 6.6 represents the statistics of the RLNILM agent performance on detecting events for different appliances.

Table 6.6 The RLNILM agent performance on appliance event detection in S2 scenarios

	Fridge		AC		Iron		Water filter	
Input signal	TP	FN	TP	FN	TP	FN	TP	FN
P	79	17	38	8	15	7	46	18
Q	75	21	40	6	15	7	44	20
I	67	29	34	12	16	6	49	15
PQ	88	8	46	0	18	4	55	9
PI	81	15	42	4	19	3	58	6
QI	80	16	39	7	17	5	52	12
PQI	73	23	38	8	16	6	50	14

By calculating the accuracy using 6.4, we have the Table 6.7.

Table 6.7 The RLNILM agent accuracy on appliance event detection in S2 scenarios

Input signal	Appliance detection accuracy			
	Fridge	AC	Iron	Water filter
P	0.82	0.82	0.68	0.71
Q	0.78	0.86	0.68	0.68
I	0.69	0.73	0.72	0.76
PQ	0.91	0.99	0.81	0.85
PI	0.84	0.91	0.86	0.90
QI	0.83	0.84	0.72	0.81
PQI	0.76	0.82	0.72	0.78

Figure 6.9 illustrates the event detection accuracy of RLNILM for each appliance with different input streams.

We observe that the AC has the highest accuracy with all input streams. It was expected as the AC has significant rising and falling edges and most event detection algorithms are able to identify that, thus, even randomly-chosen outputs among all three event detection algorithms are still informative for the RLNILM agent. Same as in the S1 group of scenarios, single inputs have lower accuracy than dual streams, and the triple stream has achieved low accuracy for all the appliances. However, in the S1 group, the accuracy scores of all the input streams were closer while in this group the difference is more obvious probably due to the equal confidence level of all the event detection algorithms.

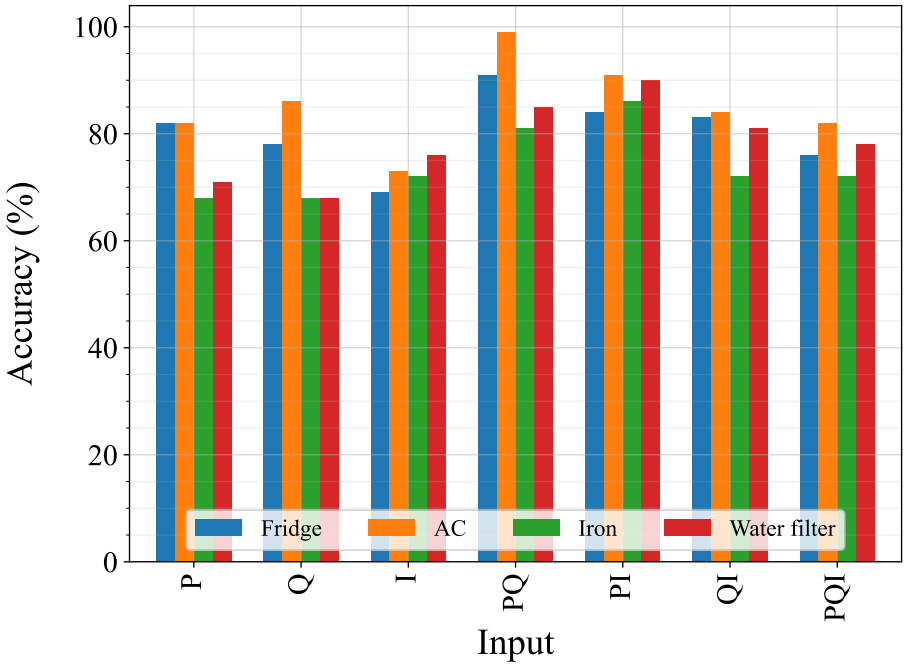

Figure 6.9 The RLNILM agent accuracy of detecting events of different appliances in S2 scenarios

6.4.3 Scenario group: S3

In this group and the next group of scenarios, An important change is applied. The input stream frequency is decreased from 1 Hz which is the main frequency of our dataset, to 0.2 Hz. It means that there is only one measurement out of every five samples that we used to have in previous scenarios. Even though the event detection algorithms in the feedback system, are optimized with 1 Hz frequency, they are still able to perform event detection with lower resolution input streams, and of course, their performance achieves lower scores than previous higher frequency cases. We kept the ground truth data (key vector) as in the previous scenarios (1 Hz resolution). It means that the RLNILM agent and event detection algorithms observe the same value as their input for 5 seconds while there may be an event happening during these 5 seconds. All the algorithms here are losing many informative load fluctuations and it adversely affects their performance, However, the RLNILM agent has the opportunity to learn from the different signal levels at those moments. This setting provides us with an opportunity to observe how much the performance score of each event detection algorithm drops when they face lower data resolution. In this group of scenarios, we keep the confidence level of event detection algorithms according to their $F_1 scores$ which helps the RLNILM to learn better.

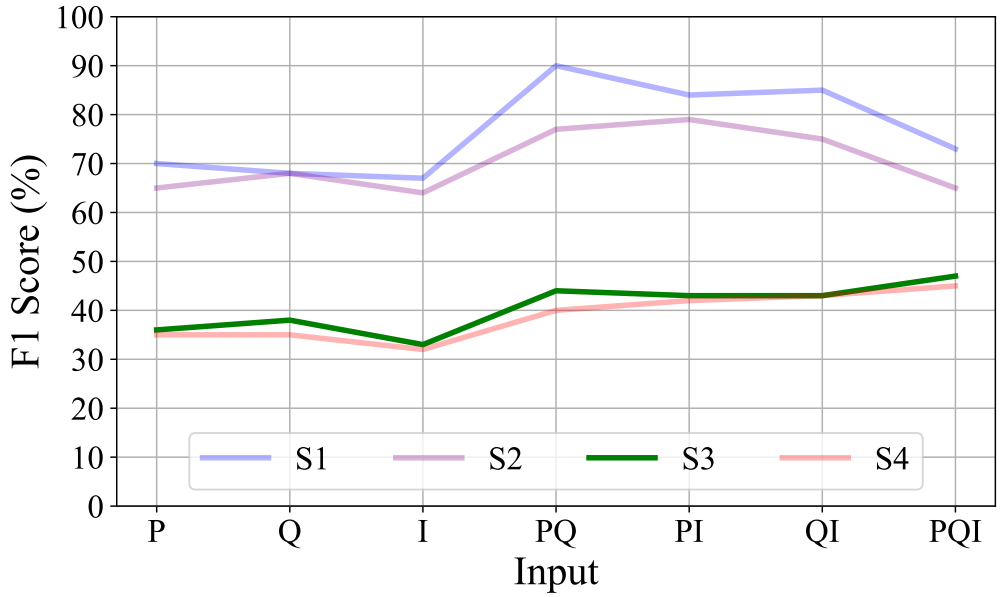

Figure 6.10 The $F_1 score$ of the RLNILM agent for different input streams in scenario group S3

Figure 6.10 illustrates the performance of the RLNILM agent with different input streams. A significant drop in $F_1 score$ of the RLNILM agent for all the inputs is obvious. As it is mentioned earlier, it is not unexpected, because a lower frequency of input data leads to a considerable loss of information for the learning process of the RLNILM agent (and all other event detection algorithms). In addition, the lower sampling rate causes the elimination of many fast load transitions, i.e., any appliance usage shorter than five seconds (sampling period) would probably have no visible impact on the load profile. For instance, most water filter usages are short, and there is a low chance of capturing them in one of the samples. On the other hand, some loads, e.g., the iron have high edges in their load signature, however, they have a short time span and they would be ignored because of the lower sampling frequency. From Figure 6.10, it is observed that there is not much difference between the performance of single stream inputs. Dual inputs perform slightly better and the triple stream of PQI is the best in this scenario. The reason that the triple input stream is working better than the dual streams (in contrast with 1 Hz scenarios) is probably the scarcity of information in the low-frequency scenarios. It is plausible to assume that in lower frequencies the amount of information that the RLNILM agent is taking out of dual streams is so low, and when the third stream gets involved, it becomes better at predicting events. It is in contrast with the case in previous scenarios (1 Hz frequency) where the involvement

of the third signal resulted in the confusion of the RLNILM agent. It is important to see how other event detection algorithms in the feedback system are performing in this scenario group versus the RLNILM agent range of results. Figure 6.11 represents this comparison.

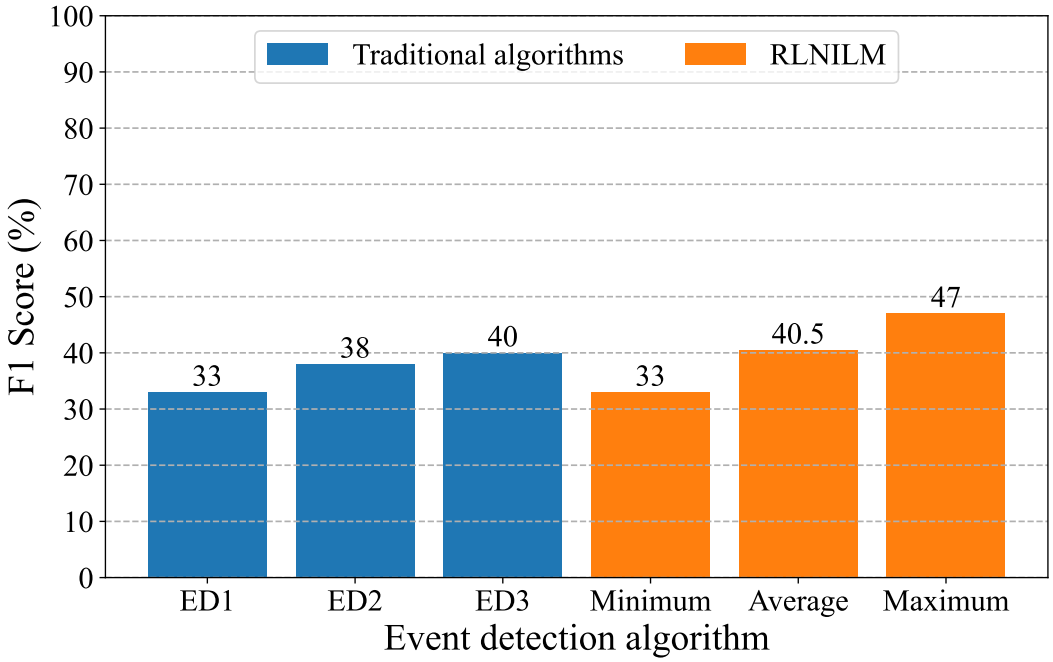

Figure 6.11 The minimum, average, and maximum $F_1 score$ of the RLNILM agent compared with event detection algorithms in the feedback system for scenario group S3

Although the focus of this study is not to analyze the performance of the event detection algorithms in the feedback system, it is still interesting to see how all the event detection algorithms have lost many of their previously-detected events (mostly on appliances with short activity spans, e.g., iron and water filter). However, the ED3 algorithm which uses both P and Q inputs to identify events, has still a higher performance than ED1 and ED2, and it is possibly due to its ability to avoid FP events because of its dual-stream input. In Figure 6.11 it is observed that the minimum performance of the RLNILM agent is almost the same as the minimum of the event detection algorithms. The average performance of the RLNILM agent is as high as the best event detection algorithm in the feedback system. Finally, we see that the RLNILM agent with the best-performing input has surpassed all the event detection algorithms. The performance of the RLNILM agent on the detection of each appliance is as Table 6.8.

Table 6.8 The RLNILM agent performance on appliance event detection in S3 scnearios

	Fridge		AC		Iron		Water filter	
Input signal	TP	FN	TP	FN	TP	FN	TP	FN
P	57	39	29	17	3	19	2	62
Q	55	41	26	20	3	19	4	60
I	49	47	25	21	2	20	3	61
PQ	60	36	31	15	5	17	5	59
PI	56	40	27	19	3	19	4	60
QI	57	39	29	17	4	18	4	60
PQI	65	31	34	12	7	15	7	57

We can calculate the accuracy table from the information in Table 6.8 which is represented in Table 6.9.

Table 6.9 The RLNILM agent accuracy on appliance event detection in S3 scenarios

Input signal	Appliance detection accuracy			
	Fridge	AC	Iron	Water filter
P	0.59	0.63	0.13	0.03
Q	0.57	0.56	0.13	0.06
I	0.51	0.54	0.09	0.04
PQ	0.62	0.67	0.22	0.07
PI	0.58	0.58	0.13	0.06
QI	0.59	0.63	0.18	0.06
PQI	0.67	0.73	0.31	0.10

These accuracy metrics of different appliances are easier to compare in Figure 6.12.

Figure 6.12 clearly shows the drop in the event detection accuracy of the iron and water filter in this scenario. However, the AC and fridge are detected at an acceptable rate due to their load signature attributes. Their load signatures have high rising and falling edges which are easily distinguished and their activity time span is detectable even at a lower frequency. It is also noted that the accuracy rate of the AC or fridge detection is still lower than their accuracy in scenarios with a 1 Hz-sampling frequency which shows that lower sampling frequency hurts all different load signatures but in different scales.

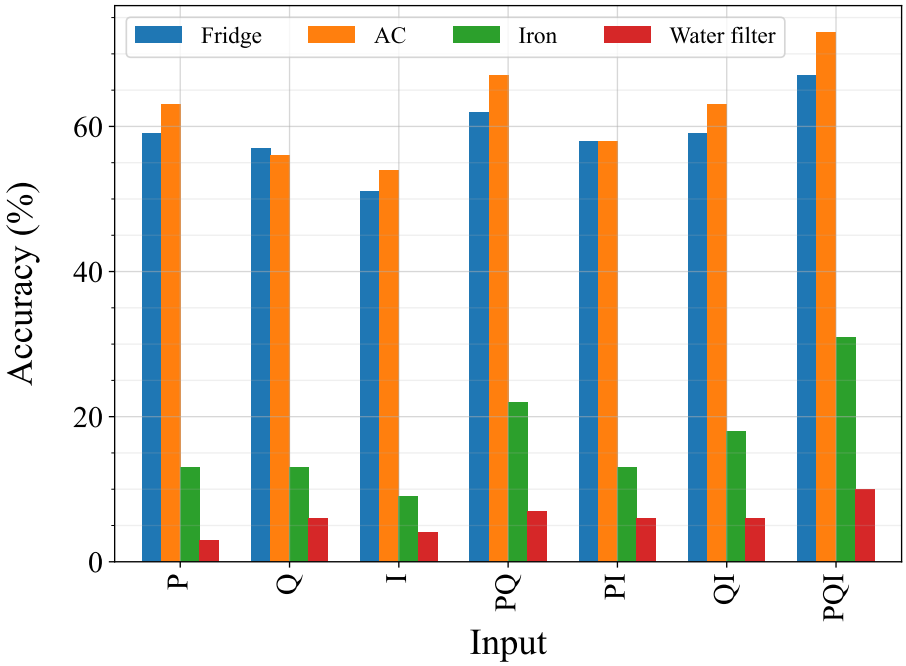

Figure 6.12 The RLNILM agent accuracy of detecting events of different appliances in scenario group S3

6.4.4 Scenario group: S4

As the last group of scenarios, the sampling frequency is kept at 0.2 Hz. We also set the confidence level of all the event detection algorithms in the feedback system equal. This change eliminates the advantage of knowing the performance score of each event detection algorithm in the feedback system which usually leads to a better learning process for the RLNILM agent. In fact, this scenario is the most difficult one compared to the three previous scenario groups.

Figure 6.13 represents the $F_1 score$ of the RLNILM agent being fed with different input streams in this scenario. If we compare this curve with the one in the previous scenario (S3) it is observed that they almost match. There are slight differences in the $F_1 score$ of the same input streams in both figures. However, the overall performance of the RLNILM agent in scenario group S3 seems better than this one (S4). It can be concluded that in low frequencies, the loss of information is affecting the RLNILM agent performance to the extent that the confidence level can not compensate for that. It means that for the RLNILM agent in this situation, it does not make any difference which event detection algorithm is giving it feedback. Thus, for this reason, it is observed that the $F_1 scores$ of different input streams

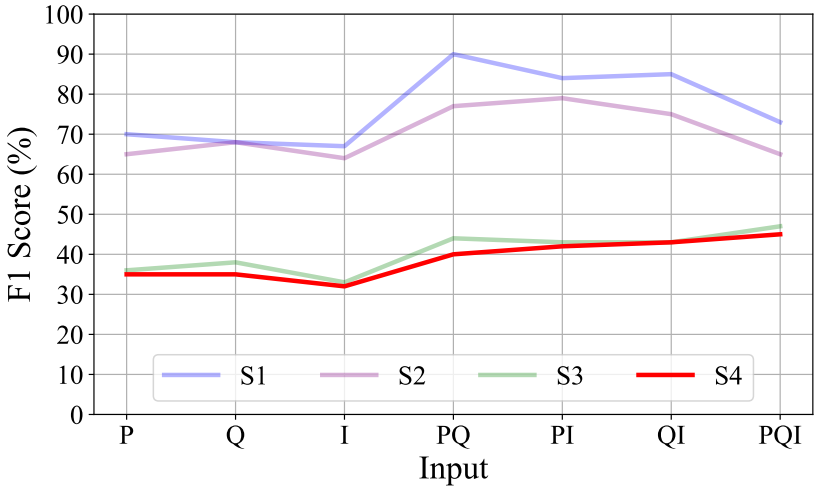

Figure 6.13 The F_1score of the RLNILM agent for different input streams in scenario group S4

are very similar to those of the previous scenario. We can see how it performs versus event detection algorithms in the feedback system by looking at Figure 6.14.

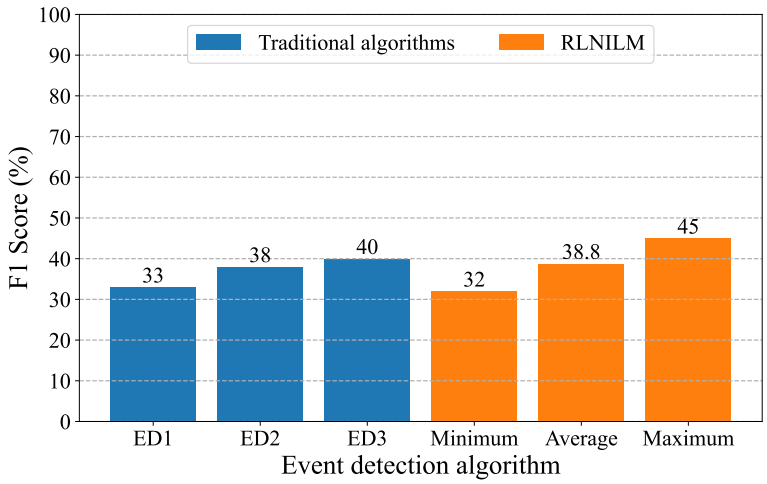

Figure 6.14 The minimum, average, and maximum F_1score of the RLNILM agent compared with event detection algorithms in the feedback system for scenario group S4

In this case, the minimum performance of the RLNILM agent is almost as bad as the worst event detection algorithm in the feedback system. The average sub-scenario also could not exceed the performance of the event detection algorithm in the feedback system. However, there exists a situation in which the RLNILM agent managed to outperform all the event detection algorithms, despite the complexity of this scenario.

As in other scenarios, we compare the performance of the RLNILM agent in detecting the events of different appliances while being fed by different input streams. Table 6.10 indicates the number of TPs and FNs of the RLNILM agent's output for each appliance in our test.

Table 6.10 The RLNILM agent performance on appliance event detection in S4 scenarios

	Fridge		AC		Iron		Water filter	
Input signal	TP	FN	TP	FN	TP	FN	TP	FN
P	55	41	27	19	4	18	3	61
Q	51	45	25	21	2	20	2	62
I	49	47	27	19	4	18	3	61
PQ	57	39	30	16	5	17	6	58
PI	53	43	28	18	4	18	4	60
QI	53	43	26	20	4	18	6	58
PQI	62	34	33	13	6	16	6	58

Using Table 6.10, we can calculate the accuracy of the RLNILM agent's performance on detecting events of each appliance as in table 6.11.

Table 6.11 The RLNILM agent accuracy on appliance event detection in S4 scenarios

Input signal	Appliance detection accuracy			
	Fridge	AC	Iron	Water filter
P	0.57	0.58	0.18	0.04
Q	0.53	0.54	0.09	0.03
I	0.51	0.58	0.18	0.04
PQ	0.59	0.65	0.22	0.09
PI	0.55	0.60	0.18	0.06
QI	0.55	0.56	0.18	0.09
PQI	0.64	0.71	0.27	0.09

Figure 6.15 compares the accuracy of the RLNILM agent for all appliances with different input streams. Same as in the previous scenario, it is observed that low sampling frequency caused information loss for some load events that have short activation time spans, e.g., iron and water filter. However, the RLNILM agent performance accuracy for all four different appliances is not much different than the previous scenario in which we had more accurate confidence level values. It seems that lowering the sampling frequency has a dominant effect that overshadows the effect of the confidence level in the algorithm.

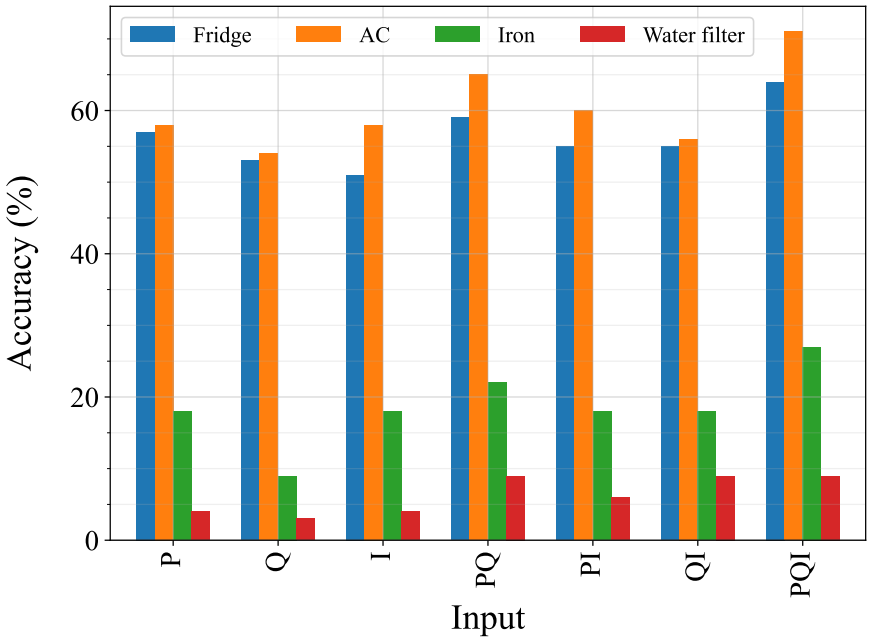

Figure 6.15 The RLNILM agent accuracy of detecting events of different appliances in scenario group S4

6.5 Discussion of the results

So far, the performance of the RLNILM agent under four different scenarios is investigated. The main scenario is the S1 scenario group in which the sampling frequency is set to 1 Hz and the confidence level is equal to the $F_1 score$ of different event detection algorithms. The sampling frequency of the main dataset (iAWE) is also 1 Hz and because the ground truth data of the dataset load events is available, we have access to the $F_1 score$ of each event detection algorithm in the feedback system. Because of these reasons, the first group of scenarios is considered as the main one. Among the scenarios of this group, the test with PQ input is considered as the most reliable result for our proposed RLNILM method, because in this case it only has access to the input streams that event detection algorithms in the feedback system are using. Thus, the scenarios where we have a 1 Hz-sampling frequency, $F_1 score$ as the confidence level, and PQ as the input stream is the most realistic scenario for testing our proposed RLNILM method. The confusion matrix of the RLNILM agent in this specific scenario is observed in Figure 6.12.

From Table 6.12, it is noted that in the main scenario, there are 216 TPs, 37 FPs, and 12 FNs which results in the $F_1 score$ of 90%. By reviewing Figure 6.5, we see that $F_1 score$ of other event detection algorithms are 78%, 60%, and 64% respectively. Thus, the RLNILM

Table 6.12 Performance matrix of the RLNILM agent performance in case of PQ input stream in S1 scenario group

	Prediction	
Ground Truth	**Event**	**No-Event**
Fridge ON	46	2
AC ON	22	0
Iron ON	10	1
WF ON	30	2
No event	37	86135
Fridge Off	46	2
AC Off	23	1
Iron Off	10	1
WF Off	30	2

method is capable of learning the knowledge of these event detection algorithms despite their differences in working principles or input requirements. Not only the RLNILM agent learned their knowledge, but also it improved them by at least 12%. At first look, twelve percent may not seem a significant improvement, however, considering the fact that the RLNILM method does not impose any criteria or requirements, makes it a solution that is worth investing in. Even in the worst scenarios of this group, it is observed that the RLNILM agent is able to combine the knowledge of three different algorithms in one single model and it performs better (score of 67%) than two of them (scores of 64% and 60%). Remember that, in general, the RLNILM agent does not need access to the ground truth data. We only used the ground truth data for evaluation purposes. In fact, the RLNILM agent is training itself through many interactions with the traditional event detection algorithms to learn how they correctly detect an event. Considering that those traditional event detection algorithms make mistakes, and the RLNILM agent is still able to learn from them shows the power of reinforcement learning.

In the second scenario, we eliminated an important advantage of the RLNILM agent which is knowing the confidence level of each event detection algorithm. Without this information (CL), the RLNILM agent is choosing its reference randomly at each time step. As some event detection algorithms make more mistakes in identifying a specific group of appliances, these mistakes will be transferred to the RLNILM agent learning process. In Figure 6.8 we see the RLMNILM agent $F_1 score$ versus the event detection algorithms in the feedback system. In the best scenario of this group, the RLNILM agent's overall score (79%) is slightly better than the best traditional event detection algorithm in the feedback system (78%), however, with the worst input stream its score (64%) is still better than two of event

detection algorithms in the feedback system. It clearly shows the importance of confidence level for the RLNILM agent learning process. For example, with PQ as input streams, the F_1score of RLNILM agent has dropped from 90% (with CL= $F_1scores$) to 77% (with equal CL for all) which is 13% decrease in performance. As it is explained before, it is not necessary to calculate the F_1score of each event detection algorithm with the dataset. It can be done once with a labeled dataset (e.g., our artificially-made iAWE dataset) and it provides us with an acceptable estimation of event detection algorithms' performance level.

In the third scenario group, the RLNILM agent is challenged more, by lowering the data resolution from 1 Hz to 0.2 Hz. We knew this fundamental change affects the performance of all event detection adversely as they are fine-tuned by the assumption of being fed with 1 Hz data. However, the goal is to test how robust the RLNILM learning ability is. In this scenario, the event detection algorithms' confidence level is set as their F_1score. In Figure 6.11, it is observed that all the event detection algorithms and also RLNILM agents' performance scores have dropped significantly, which was expected. As the input resolution decreases, many small load events are eliminated from the load profile and no algorithm can observe them, and for that reason, all the event detection algorithms are experiencing lower $F_1scores$. Despite the low-resolution data, the RLNILM agent managed to achieve a higher performance score (47%) than all three event detection algorithms (40%, 38%, and 33%). On the other hand, even in the worst case, RLNILM's performance score (33%) is not lower than the worst event detection algorithm. It is true that F_1score of 47% is not acceptable for an event detection process, however, the fact that RLNILM agent is still able to learn from this event detection algorithms with such low $F_1scores$, and even improve them, is quite amazing. In this scenario, the RLNILM agent improved F_1score of the best-performing event detection algorithm by 7% while in the first scenario group (with 1 Hz frequency) this improvement was 12%. It shows that RLNILM is able to improve the event detection function, regardless of the situation, however, the amount of improvement is dependent on the complexity of the situation. When the resolution is high enough, RLNILM improves the event detection process at a higher rate compared to low-frequency data as input.

Lastly, in the fourth scenario group, the algorithm is exposed to the most difficult situation which is low-frequency input data and lack of confidence level of event detection algorithms. Interestingly, the RLNILM agent performance scores are almost the same with its best and worst input streams (47% and 33%). It shows that the confidence level information is not helping the RLNILM agent learning process when the input data sampling frequency is low. One reason for this may be the fact that the $F_1scores$ are too low to be informative for the RLNILM agent learning process, and in fact, using $F_1scores$ in this scenario is the same as randomly choosing the output of the event detection algorithms in the feedback system.

Overall, the proposed RLNILM method shows significant improvement in the event detection algorithms which are embedded in the feedback system. It is true that the results vary in a range by changing the input streams, however, the fact that this method is capable of improving $F_1 score$ without imposing extra requirements or training, means that we can use this method to train versatile event detection models that can be used more extensively and they are not limited to specific training dataset or input signals. In the next chapter, the summary and the conclusion of this work are presented. We also discuss ideas that can be further explored in future works about RLNILM methods.

CHAPTER 7 CONCLUSION

In this thesis, it is shown that the proposed RLNILM event detection algorithm is able to improve traditional event detection algorithms under different test conditions. The proposed architecture is not imposing extra requirements on the existing systems, therefore, it can be implemented in power grids in which event detection systems are needed. The main contributions of this thesis can be summarized as follows:

- Proposing a new architecture based on the RL that can learn the knowledge of existing event detection algorithms and outperform them,
- proposing the Dual Replay Memory (DRM) for the RL structure which makes it possible for the RL algorithm to overcome complex situations e.g., NILM problems,
- Utilizing RL for the first time in a load monitoring problem that makes it more flexible and extensive in real-world cases.

7.1 Summary of chapters

In Chapter 1, the importance of having a load monitoring system in power grids is discussed. The fundamentals of NILM are explained and the different methods that are being used in NILM algorithms are discussed. Then, a specific phase of NILM algorithms which is event detection is explained. Event detection methods, e.g., expert heuristic, probabilistic, or matched filter algorithms are investigated.

Data is the most important part of every research or experiment that involves machine learning. In Chapter 2 different datasets that are being used by NILM researchers and their attributes were introduced. Then, it is explained why the iAWE dataset is chosen, a real-world dataset from a household in India. The sampling frequency of this dataset is 1 Hz and it includes many electric signals, e.g., P, Q, I of some home appliances, e.g., an air conditioner, a fridge, a water filter, and an iron.

In Chapter 3 we went through the details of the design process of two types of event detection algorithms that are used in the next chapters. Log likelihood ratio detector (LLR) which is a probabilistic method and sliding window distribution change which is an expert heuristic event detection method.

In Chapter 4 technical details are discussed. In this chapter, the use of machine learning in the NILM is introduced. The structure and use of supervised and unsupervised methods,

as the traditional machine learning methods, were discussed. After these two traditional machine learning topics, we introduce reinforcement learning which is one of the most powerful machine learning algorithms so far. Reinforcement learning is the core algorithm of the proposed architecture.

In Chapter 5 the concept of using a reinforcement learning structure to solve an event detection problem is thoroughly explained. The concepts of action, state, and reward are explained regarding the event detection problem. It is discussed how the traditional event detection algorithms are embedded in a feedback module to train the RLNILM agent. Also, a DRM module is introduced which enhances the RL algorithm performance significantly.

Finally, in Chapter 6, the RLNILM agent is put to the test to see how it performs versus the embedded event detection algorithms in its feedback system. Evaluation metrics are introduced, F_1score for the event detection performance and the accuracy for the event type identification. Four main scenario groups are defined based on the characteristics, e.g., input streams of the algorithm, input streams sampling frequency, and the confidence level of the embedded event detection algorithms.

The analysis of the results shows that in the main scenario, the RLNILM is perfectly capable of outperforming the performance of the event detection algorithms in the feedback system by at least 12%. The main part of this improvement comes from the fact that the RLNILM agent is learning the knowledge of all these three event detection algorithms, therefore, it is making fewer mistakes, and thus, the overall F_1score improves significantly. In this scenario, the RLNILM agent relies on the knowledge of each event detection algorithm based on their F_1score. It means that an event detection algorithm with a higher F_1score is more trusted by the RLNILM agent for learning purposes. In the next scenario, we removed the advantage of having access to $F_1scores$ of the event detection algorithms. As a result, the RLNILM agent's performance score dropped considerably, while it was still better than the event detection algorithm in the feedback system. In the third and fourth scenarios, the input data resolution was reduced five times. It affected all the event detection algorithms and the RLNILM agent notably. The $F_1scores$ almost halved, however, the RLNILM agent was still capable of improving the performance of the event detection algorithm in many cases. The same results happened in the last scenario where the access to $F_1scores$ as the confidence level was denied. In fact, the last two scenarios were designed just to test the flexibility and robustness of the RLNILM agent's ability to learn from the event detection algorithms and improve them.

From these results, it can be concluded that the RLNILM method is capable of learning the knowledge of other event detection algorithms efficiently, regardless of the condition of

the problem. This ability to combine the knowledge of non-perfect algorithms and improve their overall performance makes the RLNILM a versatile structure to create more powerful algorithms from the existing algorithms. This RLNILM event detection model can now be applied to different load profiles that consist of different load event types, and the RLNILM agent would be able to identify them based on the knowledge of those single-purpose event detection algorithms.

7.2 Future research

Despite all the benefits of the RLNILM method for improving the event detection process, it has some shortcomings and limitations that should be considered.

- Some event detection algorithms may not be matched with the RLNILM structure. There are some methods for event detection that may need to scan the entire load profile of a day, and then identify the load events in that. Because of this feature, they can not fit into the feedback system of the RLNILM structure. The RLNILM structure is compatible with the event detection algorithms that have real-time characteristics in them, i.e., they read the load profile one timestep at a time. This characteristic of the RLNILM method can be considered a positive point in some way, as real-time detection is of great importance in the NILM industry.

- Despite some probabilistic event detection methods, the RLNILM agent needs so many iterations to learn what it is expected to do. Because of this iterative learning process, most RL algorithms are considered computation-intensive algorithms. Thus, the RLNILM algorithm should be implemented on a central processing unit and not the edge which imposes the cost of data transfer to the entire NILM operation. It is worth mentioning that some NILM solution providers detect the load events on their device (on the edge) which is installed in the customer's electricity box and then send the detected events data for further analysis to their servers. It saves a tremendous amount of data transfer and processing costs.

- Due to the nature of RL algorithms, if a specific situation is not happening in the environment frequently, that event (state) may not be learned by the RL algorithm. In this research, we tried to alleviate this problem by designing the dual replay memory which highlights the importance of less common events for the RLNILM agent. However, in real-world cases, it is a probable problem. It is not just a problem exclusive to RL solutions, every machine learning solution which needs training is impacted by this issue.

Machine learning solutions need enough samples to be trained with, otherwise, they are not able to identify their target events. Imagine the RLNILM algorithm is reading the load profiles of the entire neighborhood in which there is only one EV. This EV is only charged once a month by its owner. Even if one of the event detection algorithms in the feedback system of the RLNILM system is capable of detecting that load event, it is hard to expect the RLNILM agent to learn that specific load event with just one sample per month while there are plenty of load event samples of other appliances.

The following items can be topics for further research in this field.

- One of the most important capabilities of the RLNILM algorithm is the ability to read and use information that is not used by traditional event detection algorithms. In this research, we tested different combinations of P, Q, and I as the input streams of the RLNILM agent, while the event detection algorithms in the feedback were being fed by their expected inputs. The results showed that when the frequency is low the more data you have the better results you get, however, with higher frequency, more data may lead to confusion of the agent. One may claim that the electric signals, e.g., power, voltage, or current, to some extent, contain redundant data, which is true. For this reason, it is very important to test the RLNILM solution with some non-electrical signals, e.g., weather or occupancy data. Unfortunately, we could not find a dataset that includes weather or occupancy data for the entire year which reflects the energy consumption changes of the household occupants based on the weather or occupancy. It is recommended to create an artificial dataset, similar to what is done in this study, which includes the weather information, heating, and cooling loads to investigate the effect of this type of data on improving the performance of the RLNILM agent. It is important to know that in some areas, the heating load accounts for the majority of a household electricity cost, thus, predicting those load events are of great importance.

- The RL algorithm explained in this thesis, is doing what it is expected to do, however, it may not be the most efficient and powerful RL algorithm ever designed. Companies, e.g., DeepMind have developed powerful and efficient RL algorithms that are capable of performing very heavy and complex tasks. It is recommended to spend more time on the RL structure itself and improve its efficiency to make it faster, less compute-intensive, and more sample-efficient in order to make the RLNILM structure suitable for being implemented on edge.

- In this study we studied three main categories of event detection algorithms and implemented them in the RLNILM structure. It is very informative to check the performance

of more complex event detection methods, e.g., supervised neural networks to see how those supervised models can transfer their knowledge to the RLNIM agent. An important benefit of this approach is the fact that supervised neural networks can be used for two important tasks in the NILM solution at the same time. The first one is event detection and the other one is load event classification. It is recommended to check if the RLNILM agent is able to learn both steps from its feedback system or not. If the RLNILM is capable of doing both these steps at the same time, the importance of this structure is doubled and it becomes much more useful as it improves two main steps of the NILM solutions. To make this suggestion more challenging, one may try to teach the RLNILM agent the entire process of a NILM solution from event detection, event classification, and consumption estimation all in one study.

- As the data is the most crucial part of every machine learning experiment, try to use the combination of many datasets to simulate the situation of reading the consumption data of the entire neighborhood. If the RLNILM agent is capable of handling such a complex input stream, it will be much more useful for real-world use cases.

REFERENCES

[1] P. V. Kamat, "Meeting the clean energy demand: Nanostructure architectures for solar energy conversion," *The Journal of Physical Chemistry C*, vol. 111, no. 7, pp. 2834–2860, 2007. [Online]. Available: https://doi.org/10.1021/jp066952u

[2] UK parliament, "Climate change act 2008," 2008. [Online]. Available: http://www.legislation.gov.uk/ukpga/2008/27/pdfs/ukpga_20080027_en.pdf

[3] S. Yilmaz, J. Chambers, and M. Patel, "Comparison of clustering approaches for domestic electricity load profile characterisation - implications for demand side management," *Energy*, vol. 180, pp. 665–677, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544219310060

[4] K. Ehrhardt-Martinez, K. A. Donnelly, and J. A. Laitner, "Advanced metering initiatives and residential feedback programs: A meta-review for household electricity-saving opportunities." American Council for an Energy-Efficient Economy, 2010. [Online]. Available: https://www.smartgrid.gov/document/advanced_metering_initiatives_and_residential_feedback_programs_meta_review_household_ele_0

[5] Government of Canada, "Energy statistics handbook." Statistics Canada, Vol. Q4, Table 8.7-1, 2009. [Online]. Available: https://www150.statcan.gc.ca/n1/en/pub/57-601-x/57-601-x2012001-eng.pdf?st=-32zmfiB

[6] P. Ducange, F. Marcelloni, and M. Antonelli, "A novel approach based on finite-state machines with fuzzy transitions for nonintrusive home appliance monitoring," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1185–1197, 2014.

[7] P.-H. Li and S. Pye, "Assessing the benefits of demand-side flexibility in residential and transport sectors from an integrated energy systems perspective," *Applied Energy*, vol. 228, pp. 965–979, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261918310237

[8] G. Strbac, "Demand side management: Benefits and challenges," *Energy Policy*, vol. 36, no. 12, pp. 4419–4426, 2008, foresight Sustainable Energy Management and the Built Environment Project. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301421508004606

[9] J. Corbett, K. Wardle, and C. Chen, "Toward a sustainable modern electricity grid: The effects of smart metering and program investments on demand-side management performance in the us electricity sector 2009-2012," *IEEE Transactions on Engineering Management*, vol. 65, no. 2, pp. 252–263, 2018.

[10] K. Ehrhardt-Martinez, K. A. Donnelly, and J. A. Laitner, "Advanced metering initiatives and residential feedback programs: A meta-review for household electricity-saving opportunities," *American Council for an Energy-Efficient Economy*, vol. Technical ReportsE105, 2010. [Online]. Available: https://www.aceee.org/sites/default/files/publications/researchreports/e105.pdf

[11] K. Carrie Armel, A. Gupta, G. Shrimali, and A. Albert, "Is disaggregation the holy grail of energy efficiency? the case of electricity," *Energy Policy*, vol. 52, pp. 213–234, 2013, special Section: Transition Pathways to a Low Carbon Economy. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0301421512007446

[12] X. Wu, D. Jiao, and L. You, "Nonintrusive on-site load-monitoring method with self-adaption," *International Journal of Electrical Power Energy Systems*, vol. 119, p. 105934, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061519328911

[13] Y. Yang, J. Zhong, W. Li, T. A. Gulliver, and S. Li, "Semisupervised multilabel deep learning based nonintrusive load monitoring in smart grids," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 6892–6902, 2020.

[14] M. A. Devlin and B. P. Hayes, "Non-intrusive load monitoring and classification of activities of daily living using residential smart meter data," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 339–348, 2019.

[15] M. Ma, W. Lin, J. Zhang, P. Wang, Y. Zhou, and X. Liang, "Toward energy-awareness smart building: Discover the fingerprint of your electrical appliances," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1458–1468, 2018.

[16] G. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[17] M. A. Devlin and B. P. Hayes, "Non-intrusive load monitoring and classification of activities of daily living using residential smart meter data," *IEEE Transactions on Consumer Electronics*, vol. 65, no. 3, pp. 339–348, 2019.

[18] B. Buddhahai, W. Wongseree, and P. Rakkwamsuk, "An energy prediction approach for a nonintrusive load monitoring in home appliances," *IEEE Transactions on Consumer Electronics*, vol. 66, no. 1, pp. 96–105, 2020.

[19] A. P. Medeiros, L. N. Canha, D. P. Bertineti, and R. de Azevedo, "Event classification in non-intrusive load monitoring using convolutional neural network," in *2019 IEEE PES Innovative Smart Grid Technologies Conference - Latin America (ISGT Latin America)*, 2019, pp. 1–6.

[20] L. Xinwei, R. Zhiren, T. Bo, L. Hui, Y. Rui, and W. Haiping, "Hybrid load identification model based on grey wolf optimization algorithm," in *2019 25th International Conference on Automation and Computing (ICAC)*, 2019, pp. 1–5.

[21] S. Gupta, M. S. Reynolds, and S. N. Patel, "Electrisense: Single-point sensing using emi for electrical event detection and classification in the home," in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, ser. UbiComp '10. New York, NY, USA: ACM, 2010, pp. 139–148. [Online]. Available: http://doi.acm.org/10.1145/1864349.1864375

[22] H.-H. Chang, K.-L. Chen, Y.-P. Tsai, and W.-J. Lee, "A new measurement method for power signatures of nonintrusive demand monitoring and load identification," *IEEE Transactions on Industry Applications*, vol. 48, no. 2, pp. 764–771, 2012.

[23] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, "Improving nonintrusive load monitoring efficiency via a hybrid programing method," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2148–2157, 2016.

[24] Z. Guo, Z. J. Wang, and A. Kashani, "Home appliance load modeling from aggregated smart meter data," *IEEE Transactions on Power Systems*, vol. 30, no. 1, pp. 254–262, 2015.

[25] A. U. Rehman, T. T. Lie, B. Valles, and S. R. Tito, "Low complexity event detection algorithm for non- intrusive load monitoring systems," in *2018 IEEE Innovative Smart Grid Technologies - Asia (ISGT Asia)*, 2018, pp. 746–751.

[26] K. D. Anderson, M. E. Bergés, A. Ocneanu, D. Benitez, and J. M. Moura, "Event detection for non intrusive load monitoring," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 3312–3317.

[27] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox, "Nonintrusive load monitoring and diagnostics in power systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 7, pp. 1445–1454, 2008.

[28] C. Nalmpantis and D. Vrakas, "Machine learning approaches for non-intrusive load monitoring: From qualitative to quantitative comparation," *Artif. Intell. Rev.*, vol. 52, no. 1, p. 217–243, Jun. 2019. [Online]. Available: https://doi.org/10.1007/s10462-018-9613-7

[29] M. Zeifman and K. Roth, "Nonintrusive appliance load monitoring: Review and outlook," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 76–84, 2011.

[30] A. Zoha, A. Gluhak, M. A. Imran, and S. Rajasegarar, "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey," *Sensors*, vol. 12, no. 12, pp. 16 838–16 866, 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/12/16838

[31] M. Weiss, A. Helfenstein, F. Mattern, and T. Staake, "Leveraging smart meter data to recognize home appliances," in *2012 IEEE International Conference on Pervasive Computing and Communications*, 2012, pp. 190–197.

[32] P. Meehan, C. McArdle, and S. Daniels, "An efficient, scalable time-frequency method for tracking energy usage of domestic appliances using a two-step classification algorithm," *Energies*, vol. 7, no. 11, pp. 7041–7066, 2014. [Online]. Available: https://www.mdpi.com/1996-1073/7/11/7041

[33] Y. Jin, E. Tebekaemi, M. Berges, and L. Soibelman, "Robust adaptive event detection in non-intrusive load monitoring for energy aware smart facilities," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 4340–4343.

[34] K. Nguyen, E. Dekneuvel, B. Nicoll, O. Zammit, C. Van, and G. Jacquemod, "Event detection and disaggregation algorithms for nialm system," in *2014 International Workshop on Non-Intrusive Load Monitoring (NILM)*, 06 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:1111659

[35] D. Luo, L. Norford, S. Shaw, S. Leeb, R. Danks, and G. Wichenko, "Monitoring hvac equipment electrical loads from a centralized location - methods and field test results," *ASHRAE Transactions*, vol. 108, pp. 841–857, 01 2002.

[36] B. Wild, K. S. Barsim, and B. Yang, "A new unsupervised event detector for non-intrusive load monitoring," in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2015, pp. 73–77.

[37] M. Berges, E. Goldman, H. S. Matthews, L. Soibelman, and K. Anderson, "User-centered nonintrusive electricity load monitoring for residential buildings," *Journal of Computing in Civil Engineering*, vol. 25, no. 6, pp. 471–480, 2011. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29CP.1943-5487.0000108

[38] S. Leeb, S. Shaw, and J. Kirtley, "Transient event detection in spectral envelope estimates for nonintrusive load monitoring," *IEEE Transactions on Power Delivery*, vol. 10, no. 3, pp. 1200–1210, 1995.

[39] L. K. Norford and S. B. Leeb, "Non-intrusive electrical load monitoring in commercial buildings based on steady-state and transient load-detection algorithms," *Energy and Buildings*, vol. 24, no. 1, pp. 51–64, 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0378778895009582

[40] J. M. Alcalá, J. Ureña, and Hernández, "Event-based detector for non-intrusive load monitoring based on the hilbert transform," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–4.

[41] N. Batra, M. Gulati, A. Singh, and M. B. Srivastava, "It's different: Insights into home energy consumption in india," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, ser. BuildSys'13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1–8. [Online]. Available: https://doi.org/10.1145/2528282.2528293

[42] M. E. Berges Gonzalez, "A framework for enabling energy-aware facilities through minimally-intrusive approaches," Ph.D. dissertation, 2010. [Online]. Available: https://www.proquest.com/dissertations-theses/framework-enabling-energy-aware-facilities/docview/845717143/se-2

[43] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484 – 9, 2016/01/28. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[44] M. Etezadifar, H. Karimi, A. G. Aghdam, and J. Mahseredjian, "Resilient event detection algorithm for non-intrusive load monitoring under non-ideal conditions using reinforcement learning," *(in press) IEEE Transactions on Industry Applications*, pp. 1–10, 2023.

[45] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.

[46] M. Etezadifar, H. Karimi, and J. Mahseredjian, "Non-intrusive load monitoring: Comparative analysis of transient state clustering methods," *Electric Power Systems Research*, vol. 223, p. 109644, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378779623005333