Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
System Architecture and Networking Group

Deep Reinforcement Learning for IoT Network Dynamic Clustering in Edge Computing

Master Thesis

Tiancong Xia

Supervisors:
Tanir Ozcelebi
Merijn van Eijk
Qingzhi Liu

Eindhoven, August 2019

Abstract

Internet of Things (IoT) networks have been deployed in different applications on a large scale. They are becoming more and more dynamic and generating increasingly large amount of data. In these dynamic IoT networks, each node needs to be connected to the server. so the connectivity and the ability of configuration greatly influence the scalability of the network. To dynamically configure the IoT network, machine learning can be a suitable paradigm, but it is difficult to gather a lot of network configuration data and label all of them for supervised machine learning.

Deep Reinforcement Learning (DRL) can learn a strategy from long-term network reward, and thus it can develop a configuration strategy. DRL has been developing rapidly and has gained huge success in different domains because of the automation it brings by reducing the effort of labeling. The advantages of DRL meet the demand of the configuration of large dynamic Internet of Things (IoT) networks.

To explore how to use DRL to configure dynamic IoT network, we first define a specific dynamic IoT context with multiple clusters, in which each cluster has a server. Then we propose a clustering method not only with good network granularity but also at the same time to keep the actions of the DRL model simple. The most suitable DRL model is used for the context. To cope with the difficulty of directly porting the simulation into real-world IoT network, we propose a DRL-based Dynamic Clustering with Synthetic Data and Target Behavior Prediction method. A Long Short-Term Memory (LSTM) model is used as a target behavior predictor to predict the dynamic factor in the real-world environment. This LSTM predictor is used to generate synthetic data at the training phase in order to solve the lack of data from real-world IoT environment. It is also used in the decision making phase to assist the DRL model about the dynamic factors in the real-world IoT environment. Three connection methods of the two models are proposed to achieve the highest expectation of reward. In the end, with the solution we proposed, we improve the state-of-the-art solution by a reward increment between 15% to 30%.

Preface

This thesis project means a lot to me. I will always remember this amazing adventure, in which I met a lot of helpful and warm people and made great progress on my skills. I want to thank all of you, that have always been supporting me and being with me in this journey.

First of all, I would like to thank my mother, Han Zaijun and my father, Xia Qingwei. Thank you so much for bringing me to this beautiful world and being my first teachers. I also want to thank my supervisors. Thank you Dr. Ozcelebi for trusting me and being patient with me. I truly appreciate all your smart suggestions and your kind instructions. Thank you Dr. Liu for always being with me. I really appreciate all your support and company. You are so friendly, generous to give a helping hand, and responsible for students. Thank you Mr. Van Eijk for encouraging me and supporting me. You helped me go through the hardest time that I first started to work in a foreign company. Thank you for all your jokes, the ones I got and the ones I did not. Thank all my other colleagues, Bram, Bas, Patrick, Dorus, Henk, Snorri, Axel, Freddy, Martin, Mark and Stijn. I truly appretiate all your company and support.

Thank you my girlfriend, He Qinting for all your unconditional support and all the good time we spent.

I wish all of you a healthy and happy life. And finally, I would thank myself for accomplishing this on time and wish myself a good summer vacation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

It is expected that around 30 billion Internet of Things (IoT) devices equipped with sensors and wireless communication capabilities will be connected to the Internet by 2020 for all kinds of purposes. With this large scale IoT network, it may face serious scalability issues to have each node connected to the Internet with each node having good connection and sufficient administration [5].

Especially when faced with large-scale dynamic IoT networks, a dynamic management paradigm is important but it is not trivial. Machine learning has had great success in industries these years on building dynamic and flexible system. However, the labeling work for large-scale dynamic IoT networks is not easy. Deep reinforcenment learning (DRL) is really suitable for this case because it can get a reward from the environment by itself which can make the whole process automated. In this way, the IoT network would be like a playground or a game for the DRL model where it can explore and develop an understanding of the network resulting in a good configuration paradigm.

DRL is considered a promisingly revolutionary in the field of artificial intelligence (AI) and it represents machine learning algorithm with a higher-level understanding of the world [6]. In recent years, DRL has gained huge success in several application domains, with deep learning enabling conventional reinforcement learning to scale to more complex and different problems. For example, DRL has gained great success in game playing. The most well-know benchmark for DRL is Atari playing. In the Deepmind paper [7], several independent improvements to the Deep Q Network (DQN) algorithm are made and the performance of DRL model is optimized greatly. The proposed Rainbow algorithm which combines six optimization approaches gives the best performance an exceeds human-level performance on over 40 of 57 Atari games attempted. A second example might be robotics using DRL algorithms, which allow control policies for robots to be learned from the camera input of the environment and constant interaction with the environment.

DRL is very popular in the AI domain because it provides a possible future solution for extracting information from a tremendous amount of data. It can interact with the environment, get the knowledge from the process and in the end be able to get a good understanding of the environment. One of the biggest advantages of DRL is that DRL needs no or minimum supervision, which saves tons of effort in labeling in real-world production. Also, DRL can learn on its findings and this provides it the ability to achieve superhuman performance.

With more and more amount of data generated by IoT networks, how to organize, process and extract information from the data is becoming increasingly significant. As most IoT networks are dynamic without a fixed pattern, DRL can be very useful in the configuration, data processing and network management of IoT networks.

However, at this moment, DRL is not used in IoT networks or the data they generate very much. While supervised learning is widely utilized in the process of data analysis instead of IoT network configurations. Implementing machine learning for IoT network configurations is also

very promising as data flow from IoT networks is mostly dynamic and it is difficult to find a pattern. With well-implemented machine learning methods, according to their characteristics, the IoT network configurations can be more flexible to different network conditions and can still make relatively advantageous decisions when confronted with situations that are never seen by the model before. However, using supervised learning is not so practical, because with the big data and fast-changing dynamical situation of an IoT network it is hard to label a network condition. So DRL models seem to be a good solution for this reason.

1.2 Motivation

Together with its automation, DRL also requires more data sample to learn from. And it needs interact with the IoT network to get the data sample. Considering the data sample the DRL needs and the IoT network working characteristics, to interact with a large-scale IoT network and get sufficient data is not always realistic. There are two reasons for this:

1. DRL can be horribly sample inefficient. In the Deepmind paper [7], the most advanced RainbowDQN proposed at that time needs about 18 million frames to pass human performance in Atari games, which corresponds to about 83 hours of play experience. And including the model training time, the DRL model can learn a game in days while most humans can learn within a few minutes. The training data sample needed is extremely large for a DRL model to converge to a useful model.

2. Most IoT devices are power-constrained devices and turn into sleep mode when they are not working to save battery, and also, IoT networks often have thousands of end-points. As a consequence, providing a large amount of training data of the whole network is not always efficient.

Therefore, alternative ways which incorporate simulation and synthetic data need to be considered [8]. As IoT networks can be really dynamic and sophisticated, static simulation cannot fully reveal the network characteristics, especially the dynamic factors in the network. But for dynamic networks, if dynamic configuration and dynamic data processing could track the most important part of the moving pattern of the networks, in theory, the process would be more efficient than static approaches.

In the simulator, the DRL model can train as many time as wanted and get a close enough grasp of the IoT network so that it can be transfered to the real-world network by transfer learning with way less amount of training sample.

This method gives a way to implement DRL into the real-world system where there is only insufficient training sample. However, it also comes with its own problems. Because the training dataset is limited, the model easily overfits to the training dataset and does not know how to handle new situations at the decision making phase. Also, some DRL algorithm have the drawbacks that it does not react to consequent input well, and when there are important dynamic factors involved in the environment, the DRL algorithms tend to have poor performance.

And these problems involved lead to our research questions.

1.3 Research Questions

In this paper, we aim to develop a scheme to use DRL for IoT network configuration. To be more specific, we will use DRL to solve the dynamic clustering problem to achieve data balance.

We assume that IoT devices produce heartbeat report data and send to edge servers periodically. Some objects move inside the area of IoT network. Once these mobile objects are detected by the sensors on IoT devices, the IoT devices produce sensing data. A clustering-based data aggregation solution is used for dynamic IoT networks. There are two challenges involved in the data aggregation of IoT networks solutions using DRL.

1. **DRL Design and Clustering Method**: How to design the DRL model to best fit the model for the context of IoT networks. This consists of two parts. The first part is which clustering method to choose and how to design the actions that DRL model can take to make the actions and clustering method cooperate well. Secondly, choosing the DRL algorithm is important for the system to converge fast and get good performance.
2. **From Simulation to the Real-world System**: The model must be able to apply/transfer the training model based on simulation to the real-world systems. Applying the DRL model based on simulation to the real-world system is not straight-forward for two reasons. (I) Most existing solutions of DRL rely on simulation for training. There is always a difference between simulation and real-world system, especially when the network is dynamic. While the real-world system cannot provide enough amount of data for training. (II) In a real-world IoT system, the observed state data cannot be collected to the DRL model in real-time continuously. Because in most IoT systems it will cost a lot of communication and computing resources to collect system state data. Therefore, most IoT systems can only provide data periodically, which means there is a time gap when only part of or even no state data is collected for the DRL model. It is hard for the DRL model to train directly on IoT network. Therefore, we need a simulator to mimic real-world conditions and produce state data. The question is how to minimize the error between the real-world observation to estimation-based simulation.

To cope with these challenges, we improve state-of-the-art deep reinforcement learning (DRL) based IoT network clustering solution. The contributions of this research are as follows.

1. We improve the DRL solution in both IoT part and DRL model.
    - In the IoT part, a more flexible clustering method, Multiplicatively Weighted Voronoi Clustering, is proposed to use for clustering IoT networks. With this method, the IoT network clustering can not only easily change the position of the clusters but also the size of them so that the clusters can react flexibly to both the position and the size of the communication density in the IoT networks.
    - In the DRL part, we use a more advanced D3QN algorithm to make the DQN learning model more powerful.
2. We design an LSTM dynamic factor predictor to generate synthetic data. The LSTM model can monitor the dynamic factor in the IoT network. Together with the static factors in the simulator, the simulation environment can generate unlimited synthetic data.
3. We design algorithms to combine the LSTM prediction model and DRL model in the decision making phase so that the DRL model can operate with the support of the LSTM prediction.

1.4 Thesis Layout

This thesis is organized as follows:

- Chapter 2 presents the theoretical background and preliminaries of this study.
- Chapter 3 states the problem and discusses the related work.
- Chapter 4 introduces a set of improvements on the state-of-the-art solution and also a DRL-based Dynamic Clustering with Synthetic data and Target Behavior Prediction method is proposed.
- Chapter 5 illustrates the results of the experiments.
- Chapter 6 concludes the results and the future work is given.

Chapter 2

Theoretical Background

2.1 Introduction to Internet of Things (IoT)

The Internet of Things (IoT) is the combination of Internet connectivity to end devices. The things are the hardware involved, which can be sensors, wearable devices, movable devices and almost everything embedded with electronics and Internet connectivity [9].

Recently, IoT technologies have been developing very fast from the end nodes to the related communication protocols, from edge computing to cloud services. Related technologies have been developing rapidly as well, such as real-time analytics, machine learning, commodity sensors, and embedded systems. Because of the combination and cooperation of these technologies, the IoT domain has seen great evolution in the past years.

2.1.1 IoT System Architecture

In a simplistic view, there are three tiers in IoT system architecture, which are devices, the edge gateway, and the cloud [10].

Devices are the end nodes, and commonly with build-in communication module, they are networked in the IoT network, such as the sensors and actuators. Edge gateways are data aggregation and routing systems with functionality, such as data preprocessing, edge analytics, fog computing, securing connectivity to cloud and providing some services for the IoT network such as network configuration. The highest tier includes the cloud applications built for IoT networks. the cloud includes various database systems that stores sensor data. In most cloud-based IoT systems, the cloud tier features event queuing and messaging system that handles communication that transmits in all tiers.

There are different IoT network topologies, the basic ones of them are point-to-point networks, star networks, tree networks, and mesh networks. In this paper, we mainly focus on mesh networks because mesh topology can manages high amounts of traffic, so they are widely used in IoT network with a large amount of data to gather [11].

2.1.2 Mesh Networks

A mesh network is a non-hierarchical local network topology. While star and tree topologies are conventional hierarchical local network topologies and in them, the bridges or switches are directly linked to only a small subset of other bridges or switches. In a mesh network, infrastructure nodes (i.e. bridges, switches, and other infrastructure devices) connect to one another directly and dynamically without hierarchy. Each note connects to as many surrounding nodes as possible and they cooperate with each other to efficiently route data between servers and clients. This lack of dependency on one node allows every node to participate in the propagation of information. This makes mesh networks robust to node failure. Mesh networks can self-organize and self-configure dynamically. This reduces installation overhead. Moreover, the self-configuration ability enables

dynamic distribution of workloads to all the nodes in the mesh network, which is very significant especially when in an event there are a few nodes should fail. Even if some nodes fail, the workload can still be routed to other nodes so that the information and their duty are not interrupted. These are the reasons why mesh networks usually have more fault-tolerance and lower maintenance costs [12].

2.1.3 Edge Computing

Edge computing is a distributed computing paradigm. Unlike a centralized computing paradigm, it does not gather all the data to the central server and process them, instead, it distributes the data and computing workload to distributed devices nodes or local servers and these are called edge servers. Edge computing brings computer data storage and processing closer to the location to the required place or network. In edge computing, the computation is largely performed on edge servers [13]. Edge computing pushes applications, data and computing service from centralized points to locations closer to the user. As a consequence, with edge computing, an IoT network can expect less latency and more security. Also, the communication with the centralized data center will be reduced. In contrast to cloud computing, edge computing refers to decentralized data processing at the edge of the network.

In the context of IoT, edge computing introduces an intermediate layer in the conventional IoT-cloud computing model. There are two layers in the cloud-driven IoT environment: IoT devices and a cloud backend. While in the edge-driven IoT environment, there are three layers in total with the edge layer in the middle. Being a middle tier of the three tiers, the edge layer plays a significant role in bridging and interfacing between the cloud and the IoT networks. An edge element in this tier can be any small-size or medium-size computing entity. And the aim of this element is to provide the storage, computing, network resource and network supervision to the applications deployed to any of the three layers. The form of this entity differs based on different situations, which can be from low-power nodes to cellular base stations. These entities may be owned and operated by cloud service providers, IoT network users or a telecom operator. This depends on how much and what form of service the telecom operator or the cloud service providers want to provide [14].

Utilizing a middle layer between end networks and the cloud is already a common practice in modern network infrastructures. Conventional middle layer functionalities mainly consist of routing, network connectivity, and network-oriented operations. While edge computing as part of the functionalities is becoming more and more popular and gradually gets into use in more and more applications [15]. Especially for IoT ecosystems, IoT networks normally need a dedicated infrastructure for specific use case but at the same time to be independent of the use case in the cloud. Edge computing can meet these demands by separating the cloud and the IoT networks from the middle layer and abstract the data to have a standardized output for the interface with the cloud.

Futuremore, the presence of the edge layer can satisfy the performance requirements of the demanding IoT services mainly in terms of latency and security [16]. The edge server can not only process the data generated from the IoT network but also can help to provision the IoT network at run time. These are major improvements for an IoT network. There are two reasons. On the IoT side, the IoT nodes normally have low communication, computation, and battery resource. Thus they are not capable of carrying out the network configuration applications all the time. With the help of edge servers, IoT networks can have significantly better performance. On the cloud side, the distance makes the latency to access and fetch data too high for these run-time applications. Also, the backbone communication expense would be very high.

As a consequence, in dynamic networks, edge computing configuration is important and the most feasible way. As the middle layer, both the computation capacity and the latency are between the other two layers, which makes edge computing suitable for IoT network configurations.

2.1.4 IoT Intelligence

IoT intelligence can be offered at all three levels: IoT devices, Edge nodes and Cloud computing [17]. The need for intelligent control and decision at each level depends on the time sensitiveness of the IoT application. IoT devices have the least latency, while edge nodes are worse in latency but have better computation capacity. Cloud computing has the longest latency but has way better computation ability than either of the other two tiers.

A wide variety of machine learning techniques have been used in IoT domain ranging from traditional methods such as regression, support vector machine and random forest to the advanced ones such as convolutional neural networks, deep reinforcement learning and recursive neural network.

Ambient intelligence and autonomous control are not the original conceptions of IoT. While with the development of them respectively, on one hand, autonomous control needs less latency to and close interactions with the physical devices. On the other hand, the need for smart applications in edge devices is growing. As discussed in section 1.1, a promising approach in this context is deep reinforcement learning where most of IoT systems provide a dynamic and interactive environment [18].

An example could be given in a smart office context, there are sensors gathering environment condition data (e.g., temperature, humidity, light strength and noise) and an agent can be an IoT device or an edge server. In such a complicated and dynamic environment, conventional machine learning algorithms do not work effectively. By reinforcement learning approach, a learning agent can continuously sense the environment state (e.g., sensing office temperature), perform actions (e.g., turn HVAC on or off) and learn through the maximizing accumulated rewards it receives in long term [19]. For supervised learning, it is hard to label all IoT network data, and during the training procedure, every action needs immediate feedback. While by reinforcement learning approach, the reward can be received from long-term system feedback, one reward can be used for several serial actions. Also, reinforcement learning can use memory data set to train and also the simulation environment can be used to accelerate the learning process.

2.2 Introduction to Machine Learning

With the advent of the big data era, there is more useful information to be mined and utilized from a large amount of data. Also with the great growth of hardware storage and computation capacity, machine learning is becoming more and more popular because of its effective performance without using explicit instructions. Instead, machine learning relies on patterns and inference. It is seen as a subset of artificial intelligence and an approach to artificial intelligence (AI). Machine learning algorithms use the training data to build a mathematical model to solve a specific problem. In this way, it uses previous experience about the problem instead of developing specific instructions for performing the task, which sometimes is infeasible because of the complexity of the problem or the input scope is too large. Machine learning has been widely used in a wide variety of applications such as natural language processing, email filtering, and computer vision. Most of these applications cannot be explicitly programmed and solved.

A machine learning approach usually consists of two main phases: the training phase and the decision making phase [20]. Usually, the parameters in a machine learning mathematical model are randomly initiated. At the training phase, machine learning methods are applied to learn the system model using the training dataset so that the model gets closer to solving the specific problem gradually. At the decision making phase, the system can obtain the estimated output for each new input by using the trained model.

2.2.1 Machine Learning Models

There are different types of machine learning algorithms. The three basic paradigms are supervised learning, unsupervised learning, and reinforcement learning.

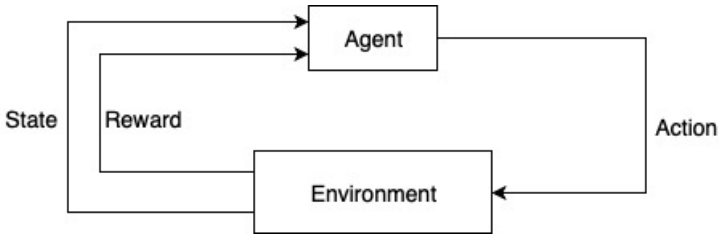


Figure 2.1: Reinforcement Learning [2]

Supervised learning is a kind of labeling learning technique. At the training phase, supervised learning uses a labeled training dataset, which means that every input has an output in the training dataset. Thus the learning model can adjust its parameters to represent a learned relationship between the input and output. After training, when a new input is fed into the system, the trained model can be used to get the expected output. Supervised learning is mainly used in spam detection, speech recognition, image recognition, robot control, and many other applications [21]. This paper involves the following supervised learning algorithms, which are neural network and recurrent neural network.

Unsupervised learning takes a set of data that only contains inputs, but it can find the hidden structure in unlabeled training data. At the training phase, unsupervised learning does not have a reward system like supervised learning to evaluate predicted output, but it finds the intrinsic structure and explores hidden patterns of the training data to divide it into similarity groups, which are also called clusters. The most common cluster algorithm for unsupervised learning includes K-means, hierarchical clustering, and hidden Markov models [22].

Reinforcement learning, as shown in Figure 2.1 is a paradigm of machine learning where learning agents use actions to interact with the environment in order to maximize cumulative reward. Unlike supervised learning, in reinforcement learning each action does not have a labeled input/output pair, and each action does not need to get the immediate greatest reward. The goal is to achieve the most cumulative reward in the end. So it has an ability of long-term strategy. To achieve this, a balance between exploration and exploitation is necessary. Exploration is the uncharted territory of the system, while exploitation is the current knowledge. Exploration can expand the current knowledge base and there is a potential to find better action with greater reward. However, it can also result in worse performance. While exploitation can achieve a known reward. This is called Exploration-Exploitation Dilemma. With exploration and exploitation, reinforcement learning can interact with the environment and develop a strategy to get good long-term reward [23].

2.2.2 Neural Network (NN)

A neural network, which is also called artificial neural network, is a software system made up of a large number of simple processing units as shown in Figure 2.2. These units are normally very simple and just have simple operations, for example, a unit could just be a linear function with a weight and a bias or non-linear computations such as the sigmoid, rectified linear unit, hyperbolic tangent functions, etc. But with a large number of these units organized, neural networks can finish complicated work [24]. The concept of it is inspired by human brains and human learning process. Human brains use a tremendous amount of neurons to perform highly complex, nonlinear and parallel computations. And infants learning through the interactions and education to gradually train these neurons to get knowledge.

In a neural network, a processing note is the equivalent component of the neuron in the human brain. Simulating the way neurons are connected in the human brain, the nodes are divided into several layers to perform similar or different functionality, and the nodes between each layer are connected to each other by variable link weights. To train the network actually is to train these weights. If the model is properly selected according to the specific problem, after enough amount

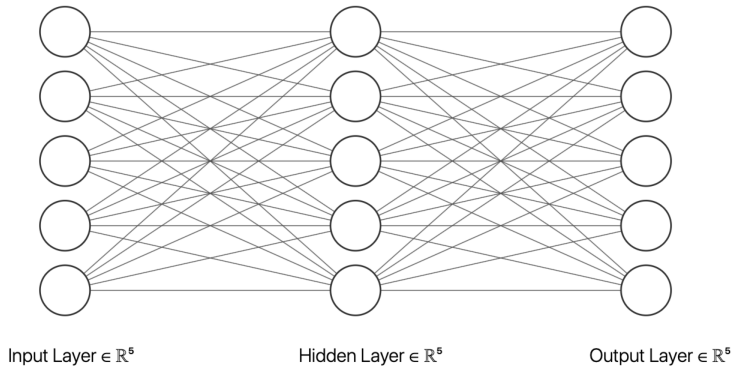


Figure 2.2: Neural Network (NN)

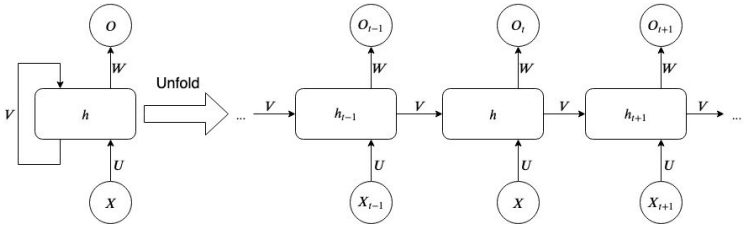


Figure 2.3: Recurrent Neural Network (RNN)

of training, the model would converge and make predictions according to the training dataset [25].

With the development of hardware computation capacity, neural networks have become the most successful methods in machine learning used in production and research. A lot of researches are being carried out with neural network and they do solve a lot of problems without programming with any task-specific rules. This makes neural networks easy to use and handy to solve complicated problems if there are enough historical data.

Feedforward neural networks are the basic and simplest neural networks in which connections between the nodes do not form a cycle. As shown in Figure 2.2, this is a feedforward neural network with three layers, i.e. input layer, hidden layer, and output layer. Each layer has five processing units. And the input size for this feedforward neural network is five and its output size is five.

There are also different upgraded classes of artificial neural network. For example, convolutional neural networks (CNNs) have achieved great success in pattern recognition as CNNs are good at finding patterns in partial areas of an image. While recurrent neural networks are widely used in speech recognition, handwriting recognition, language modeling, translation, and image captioning because they can use their internal state (memory) to process sequences of inputs, especially temporal sequences.

2.2.3 Recurrent Neural network (RNN) and Long Short-term Memory (LSTM)

A Recurrent Neural network is a class of artificial neural network where the connection between nodes forms a directed graph along a temporal sequence.

By contrast, feedforward neural networks only take input from the current time-step, while RNNs also take input or output from the last interval as one of the inputs for the current time-step. Thus, RNNs have certain abilities of memory. This allows RNNs to exhibit temporal dynamic behavior [26].

As RNNs also take input from the output of the previous time-step, they are networks with loops in them, allowing information to persist. As shown in figure 2.3, if we unfold the internal information passing into successive layers, in which each layer represents a time-step, then a basic RNN is a network of neuron-like nodes organized into successive layers, each node in a given layer is connected with a directed (one-way) connection to every other node in the successive layer. The chain-like nature of RNNs reveals that they are intimately related to sequences and lists.

However, when there are long-term temporal dependencies, RNNs have a hard time learning these dependencies because they encounter either a vanishing or exploding gradient problem [27].

These problems arise during the training of a deep network when the gradients are being propagated back in time all the way to the initial layer. The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the chain rule. And, as they approach the earlier layers, if they have small values (< 1), they shrink exponentially until they get very near 0 and make it impossible for the model to learn. This is the vanishing gradient problem, which makes basic RNNs have short-term memory and the output can be influenced more by the recent inputs than the long-term historical inputs. As a consequence, basic RNNs have poor memory and they tend to forget long-term historical inputs as they get new inputs. While on the other hand if they have large values (> 1) they get larger and eventually blow up and crash the model, this is the exploding gradient problem. These problems are inevitable for basic RNNs because of their intrinsic characteristics [3].

Long short-term memory (LSTM) was proposed to tackle these drawbacks of RNNs. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time-steps and the three gates regulate the flow of information into and out of the cell.

LSTM also receives its input from the current time-step input x_t and from the previous time-step hidden state activation a_{t-1}. While the main structural differences between the two units are:

1. LSTM introduces a memory cell state C_t.
2. LSTM also introduces three sigmoid gates, i.e. forget gate σf_t, update gate σi_t, output gate σo_t.
3. LSTM provides the abilities to remove or add information to the memory cell state [28].

In each time-step, there are several steps of an LSTM unit.

- **Step 1**: As shown in Figure 2.4, the forget gate ft acts like a switch and decides which memory cell state information from the previous time-step c_{t-1} should be discarded and which should be reserved. The mathematical mechanism is as follows. It applies a linear transformation function to its inputs h_{t-1} and x_t. Then use a sigmoid activation function to transform the output values to values between 0 and 1. These values stand for the necessity of remaining these inputs. An output value of 1 stands for that the represented input should be fully remained and passed, while an output value of 0 means that the represented input should be discarded.
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.1}$$

- **Step 2**: In Step 1, what information from the last time-step needs to be stored is determined. This step is to decide what new information the LSTM unit is going to store in the cell state. This includes two parts. As shown in Figure 2.5, firstly, a sigmoid layer called input gate layer decides which input values should be store and also which should be forgotten. This layer has the same structure with the forget gate because they are doing similar tasks and the only difference is the time-step they are used to. Secondly, the temporary memory activation $\tilde{C}_t$ is computed. The output of this step contains the temporary memory cell information

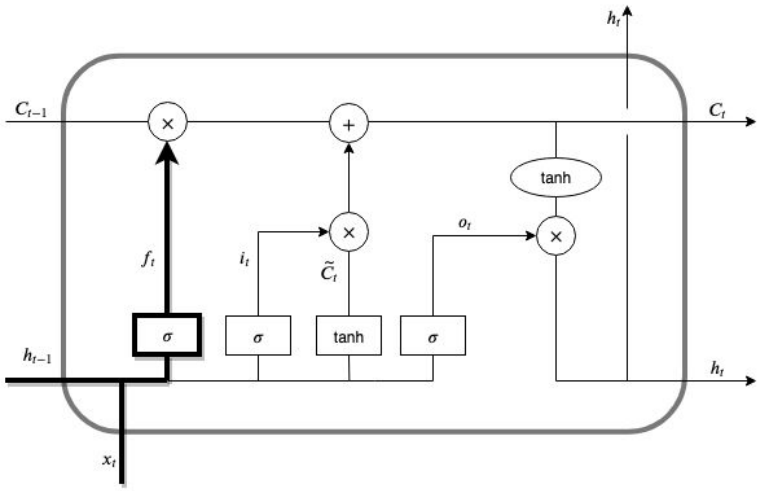


Figure 2.4: Step 1 of LSTM unit working process [3]

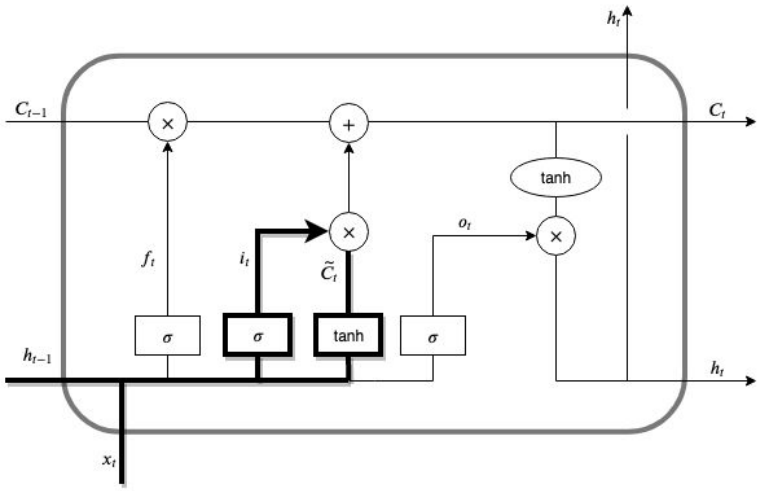


Figure 2.5: Step 2 of LSTM unit working process [3]

and it is about the current time-step. With *tanh* activation function used, the output values are between -1 and 1.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.2}$$

- **Step 3**: In this step, as shown in Figure 2.6, the cell state from previous time-step C_{t-1} will be updated into C_t. The multiplication of old cell state C_{t-1} and forget gate f_t is to forget irrelevant information. Then the multiplication of new cell state C_t and input gate i_t is used to get selected current information. Finally, the sum of them will be the new cell state C_t for the next time-step.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.3}$$

- **Step 4**: This step is used to decide the output. As shown in Figure 2.7, the output will be based on the current cell state but it needs to be filtered to output selected data. Another

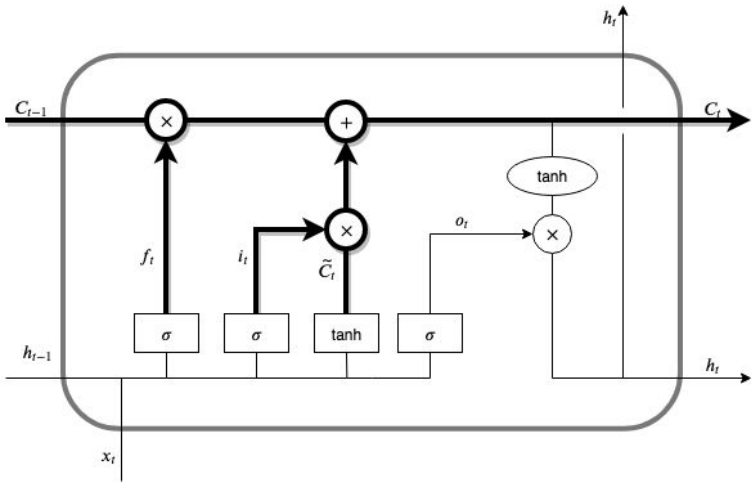


Figure 2.6: Step 3 of LSTM unit working process [3]

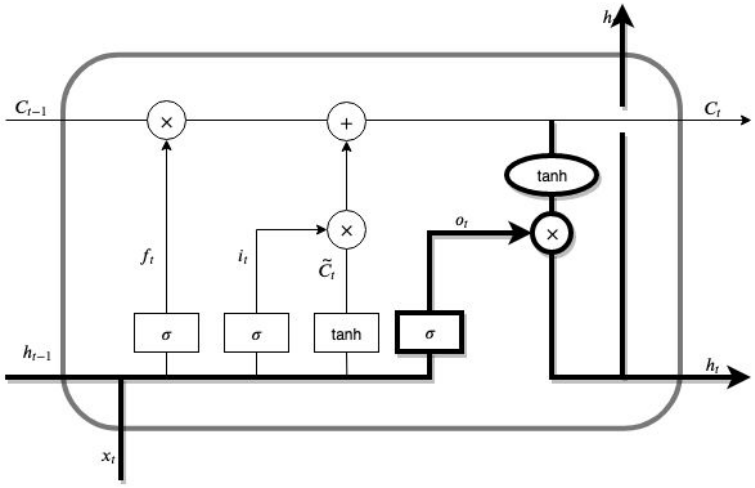


Figure 2.7: Step 4 of LSTM unit working process [3]

sigmoid layer, the output gate layer, is used to decide which parts of the cell state is useful. Then the cell state is put through a *tanh* activation function to transform the values between -1 and 1. The output of this activation function is selected by the o_t, namely the output gate layer.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) h_t = o_t * tanh(C_t) \tag{2.4}$$

2.2.4 Q-learning

Q-learning is a value-based learning algorithm in reinforcement learning. The goal of Q-learning is to learn a policy, which tells an agent what action to take under certain circumstance.

For any finite Markov decision process, Q-learning finds a policy that is optimal in the sense that the action it takes maximizes the expected value of the total reward after all the successive steps, starting from the current state [29]. As its name suggests, Q means of the quality of an action taken in a given state, and this algorithm is basically learning the Q value of each action taken in each state. The quality of an action taken in a given state represents the total accumulative

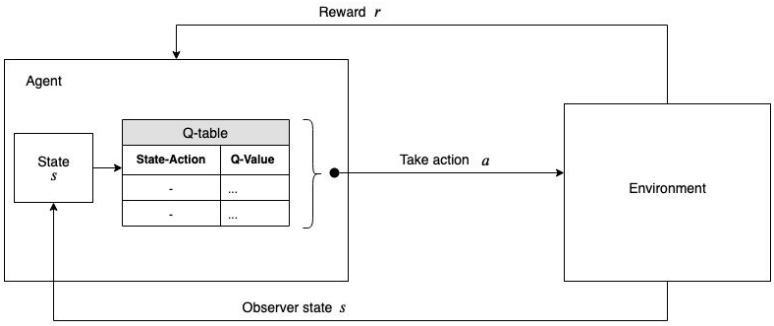


Figure 2.8: Q-learning algorithm

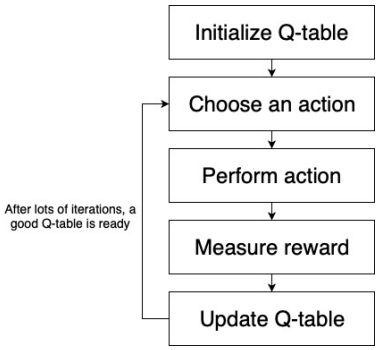


Figure 2.9: Q-learning workflow [2]

reward this action can get from this state on. Q-learning algorithm has a Q-table to maintain, in which these Q-values are listed. To train a Q-learning model is to update the Q-value in the Q-table. As shown in Figure 2.9, the Q-learning process is as follows:

- **Step 1**: Initialize the Q-table. As shown in Figure 2.8, in a Q-table there are n columns, where n equals to the total number of actions. There are also m rows, where m equals to the total number of states. Normally the Q-values are all initialized at 0.
- **Step 2**: Choose an action. Choosing an action needs to take Exploration-Exploitation Dilemma, which is mentioned in Section 2.2.1, into consideration. On one hand, exploitation helps the algorithm to use the best action that is already known and continue using it to update the Q-vales of it and the successive states. On the other hand, exploration can help explore the quality of new actions while at the same time may result in bad rewards.

  Usually, an algorithm called epsilon greedy strategy is used to tackle this. Epsilon is a possibility rate at which the algorithm will do exploration instead of exploitation [30]. The exploration means Q-learning algorithm will randomly choose any actions. While the exploitation means Q-learning algorithm will choose the action with the highest Q-value in this state (i.e. in this row).

  Usually in practice, at the beginning, epsilon is set high and Q-learning will do exploration for most of the time. The logic behind this is that at first, the RL model does not know anything about the environment. As the Q-learning algorithm explores the environment, the epsilon rate decreases and the RL model starts to exploit the environment.
- **Step 3 and 4**: Perform an action and get reward. In these steps, the Q-learning algorithm interacts with the environment. It will give the action selected with epsilon greedy strategy to the environment and the environment will return an immediate score of this action, which will be used in the next step.

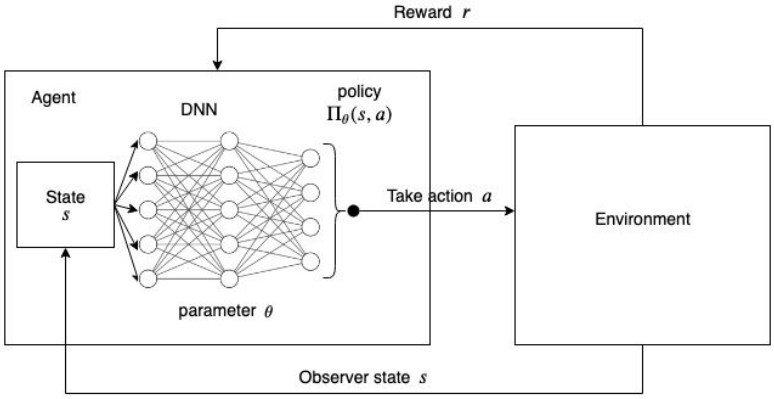


Figure 2.10: Deep Q-Network (DQN)

- **Step 5**: Evaluate the reward and update the Q-table. In this step, the Q-table will be maintained and the new Q-value will be updated. With a lot of iterations and trials of different actions under different states, the Q-table already have the experience of almost all the states and the best action to take in these states.

$$Q_{new}(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)] \tag{2.5}$$

The Equation 2.5 is the policy to update the Q-table. The aim of it is to minimize the loss, which is $[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$. In this loss part, $R(s,a)$is the current Q-value of the action in the current state, and $Q'(s',a')$ is the Q-value of the action in the next state. The maximum of this is the largest reward the agent can get in the next state. And γ is the discount factor, which illustrates how future rewards are valued. It is between 0 and 1. A bigger γ value shows future rewards are taken into more consideration, while a smaller γ value shows future rewards are less valued. A discount rate of 0 means the algorithm only cares about the immediate reward from the environment, and a discount rate of 1 means the algorithm cares every future reward equally.

$$Q_{target}(s,a) = R(s,a) + \gamma \max Q'(s',a') \tag{2.6}$$

So $R(s,a) + \gamma \max Q'(s',a')$ is Q target of the action taken in this state and $R(s,a) + \gamma \max Q'(s',a') - Q(s,a)$ is the difference between reward and the corresponding Q-value in the Q-table. And with α as learning rate, the Q-value is updated step by step.

The new value equals to the sum of original Q-value and the difference between real reward and original estimation times of learning rate. Here the real award consists of two parts, in which one part is the immediate reward from the environment, while the other part is the maximum expected future reward in the next state. As the maximum expected future reward is from the previous experience, Q-learning is an off-policy algorithm.

2.2.5 Deep Q-Network (DQN)

Q-learning is a quite effective algorithm but it is limited to problems with small amount of states. Because with the number of states increasing, it will become harder and harder to maintain or search inside of a Q-table. Moreover, Q-learning is not so smart. It can only give useful output for the state it has experienced before. When faced a new state, even if it is really similar to one state with previous experience, the Q-learning algorithm cannot realize the similarity between them and give an estimated output to the new state. So when there are so many states involved that at the decision making phase and most of the states are new to the Q-table, Q-learning algorithm would have a very bad performance. Also, deep Q-learning gives a Q-value to every action when

it receives a state as an input, while Q-learning only takes both state and action as input, and only give the Q-value for this specific action [31].

The solution of DQN is to use a deep neural network (DNN) instead of a Q-table. With the non-linear feature of neural networks, they can recognize similar states and adjust itself to the environment without enumerating every state in a table.

2.2.6 Double DQN and Dueling DQN (D3QN)

In the Deepmind paper [7], several independent improvements to the DQN algorithm are made and the performance is optimized greatly. The proposed Rainbow algorithm which combines six optimization approaches gives the best performance. Here Dueling DQN and Double DQN (DDQN) are selected to be introduced.

- **Double DQN**: Double DQN is introduced to handle the problem of overestimation of Q-values. In Equation 2.6, we assume the best action for the next state is the action with the highest Q-value. However, at the beginning of the training, there is no enough information about the best action to take. As a consequence, taking the maximum Q-value as the best action to take can lead to false positives [7, 32]. If non-optimal actions regularly have higher Q values than the optimal actions in the training phase, the Q-values will be overestimated and learning may not converge to best actions.

  To decouple the action selection from the target Q value generation, Double DQN solution uses two networks to compute the Q target.

  $$Q_{target}(s,a) = R(s,a) + \gamma \max Q(s', argmax_a Q(s',a)) \tag{2.7}$$

  First, DQN network selects the action with the highest Q-value to take for the next state. Then the target network is used to calculate the target Q-value of taking that action at the next state. As a result, Double DQN reduces the overestimation of Q-values and therefore also accelerates training process.

- **Dueling DQN (DDQN)**: Q-value $Q(s,a)$ reflects how good it is to take an action a at state s. How DDQN improves DQN is to decompose $Q(s,a)$ as the sum of:

  1. $V(s)$: the value of being at the state
  2. $A(s,a)$: the advantage of taking that action at the state rather than all the other possible actions at the state.

  $$Q(s,a) = A(s,a) + V(s) \tag{2.8}$$

  DDQN can separate the estimator of these two elements using two new streams:

  1. one that estimates the state value $V(s)$
  2. one that estimates the advantage for each action $A(s,a)$

  These two streams are combined through a special aggregation layer to get $Q(s,a)$. With DDQN, the model can learn whether the state is valuable or not without having to learn the effect of each action at each state. Sometimes, the overall Q-value for one state is small and the action choice does not influence the environment in a relevant way. And this is when DDQN especially stands out. In this case, it is unnecessary to calculate the value of each action. Thus, the model can have a better understanding of the environment. As a result, the learning can be more effective and more efficient and the performance can be better than DQN [7].

Usually, the combination of DDQN and Double DQN is called D3QN [33]. In this paper, D3QN algorithm will be used on DQN because they together provide better system performance, which at the same time will not make the system too complicated.

Chapter 3

Problem Statement and Related Work

3.1 Problem Statement

The technologies and implementations of Internet-of-Things (IoT) have been growing rapidly during the last decade, because of its capability to control, monitor and interconnect real-world objects [34].

The increasingly ubiquitous and powerful smart devices such as sensors and smartphones have been promoting the fast development of IoT applications which generate more data and require more data aggregation and processing capabilities. The applications with massive data transmission are mainly streaming applications such as augmented reality, interactive gaming, and event monitoring. The IoT applications with tremendous data production and processing have made IoT a major source of big data [35]. Currently, most IoT applications use server-client architecture with front-end smart devices and the back-end cloud. They gather the data from IoT sensors and send them to the cloud with access points assigned to certain areas of the network. However, the long-distance interactive communication between billions of end devices and the cloud may well result in two major issues: latency and capacity. On one hand, Cloud solution has typical end-to-end latency of hundreds of milliseconds and some applications require way less latency than this. On the other hand, the big data streams may not be affordable by today's network infrastructure. Moreover, in the IoT network, in which there is end-to-end communication, for example in a mesh network, many brokers and servers may be geographically distributed in the network and if there is some event happening in certain areas, both the routing process and the access point of the area may well be congested. This would result in IoT network congestion and access point overloaded.

One effective solution for this is to use the emerging edge computing architecture to offload the back-end computing tasks from the cloud to edge servers. With shorter distance to the end devices, the use of edge servers can not only reduce the backbone Internet traffic but also provide service with lower latency and better resilience than the traditional cloud paradigm.

But still, edge computing mainly solve the problem of backbone Internet overload and when there are more than one edge servers for an IoT network, there still exist two major problems with this solution for a big data IoT network with several subnetworks, which we refered as clusters:

1. **Data aggregation**: to gather data from end nodes to edge servers in the IoT network, especially for mesh networks. It is important to balance the communication load in each cluster. There are two reasons for this: one is that the balanced communication load can make power consumption more distributed to each end node; another reason is that similar communication load in different clusters can make the whole network work more efficiently overall and reduce the chance of IoT data transmission congestion.

2. **Load balancing**: In a dynamic network, each server has to process dynamically changing

amount of data load gathered in different clusters, which makes the load on cluster servers unbalanced [36]. This will result in the waste of computation capacity of less used servers and even congestion in the heavily used ones, which will increase the system latency.

To date, many solutions for data aggregation in IoT networks and load balancing for parallel computing have been proposed respectively. However, most of the techniques in the two fields are totally independent of each other, and the research on optimizing the performance on both aspects is limited. Meanwhile, most existing solutions focus on IoT network with a static data flow pattern. As the progress of IoT systems, the data aggregation scheduling should be adaptive to the dynamic networks. Together with the complexity of this problem and the difference between each IoT networks, the best solution could be machine learning solution and there is already a lot of related work carried out in this domain.

3.2 Related work

In the related work, supervised machine learning is mainly used in IoT network prediction and configuration. Also, traditional machine learning techniques such as SVM, K-means Clustering, and together with deep neural network are used in traffic classification. CNN and RNN are mainly used in mobile pattern recognition [37].

- **Network Prediction and Configuration**

  Alawe *et al.* [38] employ neural network for traffic prediction to improve network scalability. Zhang and Patras [39] build a machine learning model for traffic forecasting, and to provide the model long-term forecasting ability, they combine 3D-CNNs and ConvLSTMs. Nie *et al.* [40] use Deep Belief Network (DBN) and Gaussian models to predict wireless mesh network traffic. With the combined models, both long-term dependency and short-term fluctuations are considered. Jiang and Deng [41] propose a DRL solution for multiple-group IoT network optimization. They use DRL to determine the amount of radio resources allocated to each IoT sub-network for random access and data transmission. And the objective is to maximize the long-term average number of IoT devices that are able to access and deliver data. Liu *et al.* [1] propose a DRL solution on IoT network clustering but in simulation environment. This paper will define the problem based on their work, and optimize the state-of-the-art solutions and propose methods to port the DRL algorithm from simulation to the real-world environment.

- **Traffic Classification**

  Apart from traditional machine learning techniques, the classification neural network is also used in traffic classification. Wang *et al.* [42] use CNN to classify malware traffic from raw traffic. Abnormal traffic can be distinguished by the CNN model but the malware needs to be known or at least similar to the training data set. For encrypted traffic classification, Wang *et al.* [43] and Lotfollahi *et al.* [44] both proposed CNN-based solutions.

- **Mobile Pattern Recognition**

  In this domain, Yao *et al.* [45] use CNN and RNN for moving target tracking, human activity recognition, and user identification. Zou *et al.* [46] use cloud-based Autoencoder, CNN and LSTM for human activity recognition using commodity WiFi-enabled IoT devices.

Most of these applications in the IoT networks use supervised learning, which needs a great amount of training data. In terms of IoT configuration, IoT networks cannot generate that much data for training. Liu *et al.* [1] propose a DRL-based Dynamic Clustering and prove DRL can dynamically configure the IoT network in simulation. However, there is not so much research about porting DRL from simulation to real-world environment and how to direct the DRL model to learn the key dynamic factors in the real-world environment.

Chapter 4

System Design

4.1 Reproduction of DQN-based Dynamic Clustering [1]

This thesis is based on the work of Liu *et al.* [1], so the following problem definition is based on their problem definition. Futhermore, we derive a formula that can convert the reward defined to network throughput, which is more explicit.

4.1.1 Problem Definition

The state-of-the-art system model consists of an IoT network and edge servers as shown in Figure 4.1. The IoT network consists of homogeneous end nodes, which are randomly deployed in the area. These end nodes compose several mesh subnetworks and each end node only belongs to one subnetwork. In this way, the network is partitioned into clusters, which is for collecting data from IoT devices to multiple edge servers. Each mesh network has an edge server to make parallel computing for the data collected from the IoT network. And each end node in this subnetwork has multi-hop communication routes to the edge servers. Each edge server is responsible for collecting the data from the IoT devices of its cluster.

It is supposed that regular report data of fixed size is sent periodically by each IoT node to the edge server of its cluster. A dynamic object is moving in the area of the IoT network and triggers the nearby IoT nodes to produce sensing data of a larger size. In this way, the IoT network is dynamic in terms of data size and network density.

For example, the context could be a truck moving around in a warehouse and the IoT network is to monitor the work process of the truck. When a truck appears in the detection range of an IoT end node, the IoT device starts to capture the image of the truck and send image data to the edge server of its cluster. So to keep the data load in subnetworks balanced, the network clustering should depend on the layout of the network and the position of the moving object.

Figure 4.2 presents an example. To guarantee the collected data size in both edge servers balanced, the network clustering pattern should change to adapt to the movement of objects in the area. In the state-of-the-art research[1], a DRL approach is used to find the optimized network clustering in the dynamic IoT network. Voronoi diagram is used to control the partition of clusters [47].

As shown in Figure 4.3, to control the partition of clusters, firstly, virtual position cluster-cores are defined. Each cluster will have a cluster-core. And every end node chooses the closest cluster-core. In this way, the partition forms a Voronoi diagram [4].

To control the partition of the clusters is actually to move the cluster-cores. The action state-space A is the movements of all the cluster-cores. For example, in the implementation, each cluster-core has five movement actions: Up, Down, Left, Right, Stay. At each time step, the DQN model selects and performs a movement action on a cluster-core. After the movement, the network re-selects the IoT network node that is closest to the cluster-core o_i as the cluster head h_i,

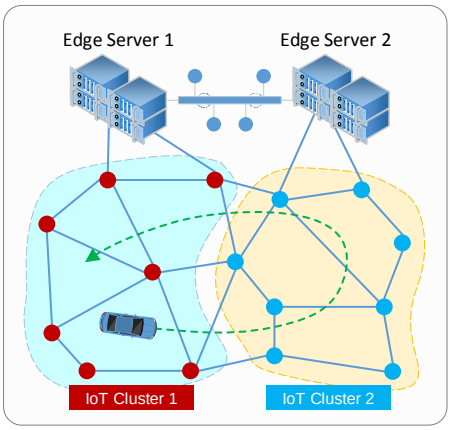
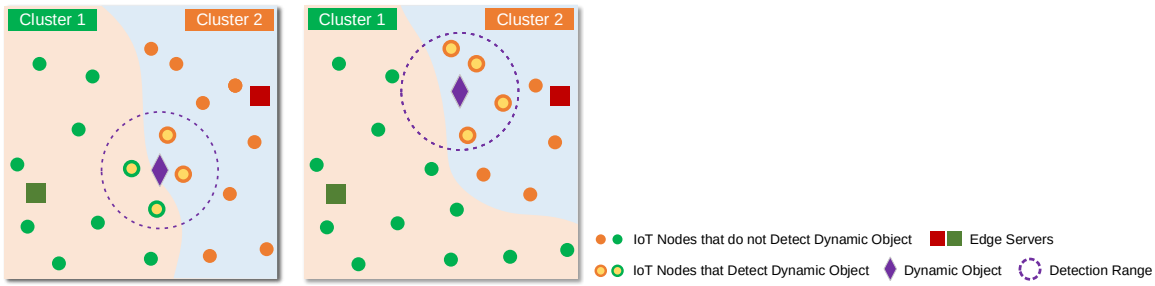


Figure 4.1: The state-of-the-art system model of IoT network and parallel edge servers. The data collection is based on the partitioned clusters.



(a) (b)

Figure 4.2: The dynamic object triggers the neighbor IoT devices to produce data. As the object moves from the position as in (a) to (b), the cluster partition changes to maintain the balance of collected data in the edge servers.

and re-partitions the network into clusters based on the new cluster head. An example of moving cluster-cores and partitioning clusters is shown in Figure 4.4.

4.1.2 The Existing DQN-based Dynamic Clustering [1]

Actions

Suppose there are a set of edge servers $P = \{p_1, ...p_i, ...p_n\}$ which reside in the network with integer $i \in [1, n]$, initially, a cluster-core o_i is set at the position of edge server p_i. During the initialization phase, to partition the IoT network into clusters, first, the IoT network node that is closest to the cluster-core o_i is selected as the cluster head h_i. Subsequently, the network can be partitioned into clusters $C = \{c_1, ...c_i, ...c_n\}$ with $i \in [1, n]$ based on the cluster heads with Voronoi diagram method (each Voronoi cell will be a cluster). After the initialization, each edge server resides in a cluster.

So to adjust the partition of the IoT network, the actions will be moving the cluster-cores, i.e. moving up, down, left, right or stay.

Reward

As in the context of parallel computing and mesh networks, both parts need to be considered. The more both data load in the edge server balances and communication densities balances in

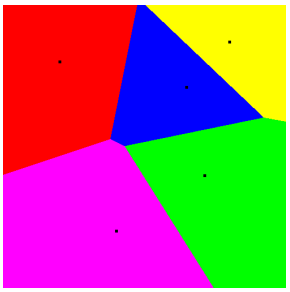


Figure 4.3: The plane is divided by the vertical bisectors of every two cluster-cores. In this way, in every Voronoi cell, the closest cluster-core for any point is the cluster-core of this Voronoi cell [4].

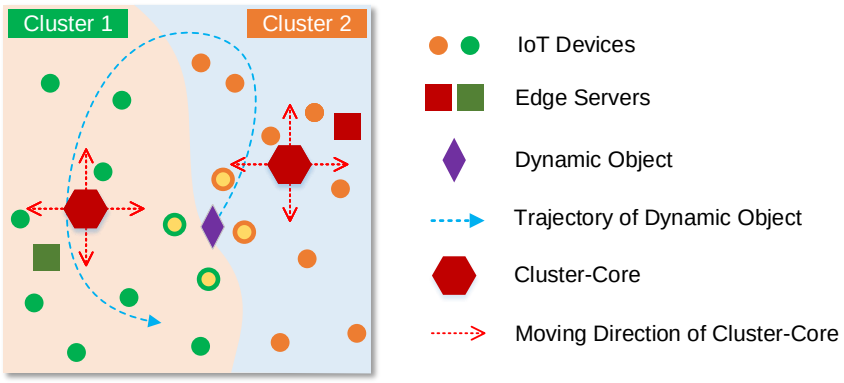


Figure 4.4: The IoT network is partitioned into clusters based on the movement of cluster-cores.

the subnetworks, the more the reward should be. As a consequence, the reward of an action is how satisfactory it is to the requirements on both IoT network and edge computing aspects. It is defined in the following form.

- IoT Networks: Name the total number of communication hops to transfer all the data in cluster c_i to edge server p_i as b_i. $B = \{b_1, ...b_i, ...b_n\}$ with $i \in [1, n]$. As it is assumed that the IoT network consists of mesh subnetworks, which uses multi-hop communication, more communication hops mean higher communication load for mesh networks to collect the same amount of data, which is less efficient. So with optimized clustering method, even with the same data amount to aggregate, the communication hops could be less and the network is less likely to get congested. To keep the speed for collecting data of each cluster on the same level and reduced the possibility of congestion in overloaded subnetworks, the number of communication hops for transferring all the data in each cluster should be balanced. The average absolute deviation value of B that is normalized in $[0, 1]$ is used to quantify the requirement of IoT network, which is defined as the communication balance index in the mesh networks H:

$$H = \frac{1}{n} \sum_{i=1}^{n} [1 - \frac{|b_i - \overline{B}|}{\overline{B}}] \tag{4.1}$$

- Edge Servers: Name the size of data collected $D = \{d_1, ...d_i, ...d_n\}$ with $i \in [1, n]$, in which each edge server p_i collects d_i. To balance the computing load in edge servers in order to maximize the overall parallel computing speed, the collected data size should be balanced in the edge servers. In order to quantify this load balancing requirement, the average absolute deviation value of D that is normalized in $[0, 1]$ is used, which is defined as the data balance

index in the servers W:

$$W = \frac{1}{n}\sum_{i=1}^{n}[1 - \frac{\left|d_i - \overline{D}\right|}{\overline{D}}] \tag{4.2}$$

To optimize both aspects of the system, the two indexes should be combined together as the reward of DQN model. The reward function at time t is expressed as $r_t = H \times W$.

The objective of DQN model is to learn a policy to maximize the expected cumulative discounted rewards R_t as follows, where $\beta \in [0, 1]$ is the discount factor for reward r_t and k is the number of time steps from t.

$$R_t = E_\pi \left[\sum_{k=0}^{\infty} \beta^k r_{t+k}\right] \tag{4.3}$$

State

Name s_t as the input state data of the DQN model at time t in state-space S, which is an array value concatenated by: s (i) the adjacency matrix of the network; (ii) the cluster ID of each node; (iii) the location of mobile events; (iv) the moving direction of mobile events. The first two are the network state, which is the static factor while the last two is the moving target state, which stands for the dynamic factor. A policy π is a mapping from the state-space S to the action space A. The DQN aims to learn the optimal policy to maximize R_t.

Q-Value

The Q-value $Q(s_t, a_t)$ is defined as the reward R_t when taking action a_t in state s_t using policy π. Q-values are used to measure the quality of a certain action in a given state. the DQN model in this case, uses a fully connected DNN with weight set θ to approximate the Q-function. After taking the action in the environment, the agent receives the reward r_t and the IoT network moves to a new state s'. At the decision making phase, We select the action that maximizes the Q-value as the one to be taken in the state s_t.

$$a_t = \arg\max_{a \in A} Q(s_t, a) \tag{4.4}$$

The update policy with Q-values is based on the following update equation, where $Q^{'}$ is the improved policy and $\alpha \in [0, 1]$ is the learning rate.

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \beta \max_{a \in A} Q(s', a) - Q(s_t, a_t)] \tag{4.5}$$

DNN is updated by the DQN model at each iteration to minimize the loss function L.

$$L = [r_t + \beta \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)]^2 \tag{4.6}$$

4.1.3 Existing Problems of the Existing DQN-based Dynamic Clustering Solution [1]

In the state-of-the-art research [1], the DQN-based dynamic clustering solution improves the reward of the static clustering by around 30% to 50% in the simulation environment, in which the moving object has a basic point to point moving pattern. The larger the cluster number is, the less the improvement DRL model provides. The main reason is that only one cluster-core is selected to move in a time step. If all the cluster-cores perform actions in a time step, the chance to find the cluster partition with higher reward would increase. The previous work proves that, in the simulation context, the DQN-based solution can be used to improve this kind of clustering problem. However, there are still existing problems for the model and there are ways to improve this solution.

Clustering Granularity

In state-of-the-art research, Voronoi diagram is used to determine the clusters of the IoT networks. So the segmentation lines are the perpendicular bisectors of the lines between every two cluster heads. These segmentation lines are straight lines. So this segmentation method cannot provide clusters with curve edges.

Furthermore, this clustering method is not so flexible considering that it can only change the positions of clustering heads. The data flow density in most IoT networks is not evenly distributed. When some event happening in one area, the network communication becomes dense around that area and more information needs to be processed. In this circumstance, this cluster would have a higher chance to suffer from network congestion. So this network needs to distribute the workload to the surrounding clusters in order to balance the data flow between each cluster. However, to shrink the area of the corresponding cluster, all the surrounding cluster heads need to move closer to share nodes for the overloaded cluster. This needs a set of actions from all the close-by clusters, which is not that efficient.

However, this is understandable because, with this simple clustering solution, there are already 5 actions for each cluster for the DQN model to choose from. The increasing complexity of the clustering method will bring better clustering granularity but at the same time may also result in longer training time or even DQN model unable to converge.

The ideal solution would be a clustering method which can not only determine the position of the cluster head but also the density of data load in the cluster. At the same time, the clustering method should also not be very complicated to have too many parameters to control. With too many parameters to control, it will need too many actions to choose from. Therefore the scale and the complexity of the model will increase dramatically.

DRL Model

DQN has a basic ability to learn from environment. However, compared with the other state-of-the-art solutions, its learning efficiency and converged performance are much lower. Therefore, to achieve higher performance, we utilize D3QN model in our system, which is the combination of DQN, Double DQN, and Dueling DQN. Compared with the other DRL solutions, D3QN is able to achieve relatively high performance and is simple to implement.

Also, the r_t in the DRL model reward equation is not intuitive. We only know r_t is lower than 1 and the closer it gets to 1, the more both the IoT network communication and the server data load are balanced. An explicit formula needs to be derived to relate the reward with the IoT network index.

Obstacles to Implementing DRL Model into Real-world Network

In state-of-the-art research, simple event movement patterns are used. They are deterministic without any noise. As a result, the training patterns are easy to duplicate and learn by the DRL model. This may also result in overfitting. However, real-world IoT network can be very complex and dynamic, so it would be hard to directly port a simulation DRL model to a real-world network. There are two main reasons for this:

- **Not Enough Training Data**

  In a real-world system, the DRL model could encounter new system state in decision making phase. The performance would drop significantly. To tackle this, the DRL model needs more real-world training data. But as discussed in Section 1.2, it could result in a long training period and high training cost if trained in a real-world network.

- **Difference between simulation environment and real-world network**

  A possible solution for the lack of training data would be to build a simulator to mimic the real-world environment. In a simulator, the training data can be synthetic thus unlimited.

But a new problem would arise in this solution. The simulator will deviate from the real-world environment more or less for sure. This will lead to bad performance.

As we know, the differences between the real-world environment and the simulation environment can lead to catastrophic performance decline for DRL model. So the trained model may not be directly used in a real-world network. So the key point here is about how to compensate for the difference between the simulator and the real-world environment. If the environment of DRL model is static and in some cases when dynamic factors also influence the performance of the environment a lot, the static environment would be far from sufficient to mimic the real-world environment. As a consequence, a dynamic factor predictor can be used to mimic the dynamic elements of the real-world environment.

4.2 Improvements on the Existing DQN-based Dynamic Clustering Solution [1]

4.2.1 Multiplicatively Weighted Voronoi Clustering

In the existing DQN-based Dynamic Clustering solution [1], Voronoi diagram is used to partition iot network into clusters., and each cluster head has 5 movement actions to adjust its position, i.e. stay, up, down, left and right. However, this method provides poor granularity as the partition edges are always straight lines and the clustering partition only depends on the positions of the cluster heads. They do not respond to the density change of each cluster.

As shown in Figure 4.5, multiplicatively weighted Voronoi can partition a plane with curves. The multiplicatively weighted distance between any point in every multiplicatively weighted Voronoi cell and the cluster-core is shorter than that between the point and any other cluster-core [48]. From Figure 4.5a to Figure 4.5d, the weight of the blue cluster is increasing, resulting in the expansion of the area.

To avoid adding too many actions and provide better granularity, two additional actions are added to change the Voronoi diagram clustering method into multiplicatively weighted Voronoi diagram clustering, which can control the multiplicative weight of each cluster head, i.e. shrinking and expanding. In this way, the model reacts more effectively to the change of density in the networks [48].

Figure 4.5 shows that in a multiplicatively weighted Voronoi diagram, if the blue part keeps increasing its weight, it will shrink, and other clusters will take its original area. So now there are 7 types of action for a cluster head, and they are to position stay unchanged, move up, move down, move left, move right, area shrink and expand.

4.2.2 D3QN

Because we implement the DRL with real-world event positions, more powerful DQN algorithm needs to be used to achieve better model performance and faster speed of convergence.

In the Deepmind paper[7], RainbowDQN achieves the best outcome with 6 improvement methods on the original vanilla DQN. In this paper, the focus is not on the DQN improvement part, therefore we use D3QN, which is good enough to have performance close to RainbowDQN at the same time maintain a simple network layout.

With Double DQN, we change the target Q-value into:

$$Q_{target}(s,a) = R(s,a) + \beta \max Q(s', argmax_a Q(s',a)) \tag{4.7}$$

Then the update policy with Q-values are based on the following update equation, where Q' is the improved policy and $\alpha \in [0,1]$ is the learning rate.

$$Q'(s_t,a_t) = Q(s_t,a_t) + \alpha[r_t + \beta \max Q(s', argmax_a Q(s',a))) - Q(s_t,a_t)] \tag{4.8}$$

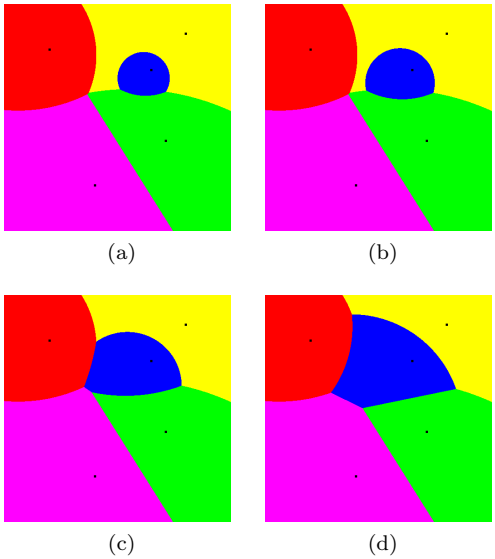


Figure 4.5: Multiplicatively weighted Voronoi can partition a plane with curves.

With Dueling DQN, the reward of being at a state and taking an action at the state is separated and Q-values are divided into these two parts, where a' is any action in state s, κ is the total number of any action a' in state s, θ is the common network parameters, α is the advantage stream parameters and β is the value stream parameters:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\beta,\alpha) - \frac{1}{\kappa}\sum_{a'} A(s,a';\beta,\alpha)) \tag{4.9}$$

4.2.3 Relation between the Reward and the System Throughput

In order to make the value of r_t or W and H more explicit, we define the throughput of the servers as T, which describe how much data all the servers are albe to process in certain short period of time, in which the network configuration and the amount of data generated from each node do not change. And we do not allow any conggestion in the servers. So no server should exceed the maximum data procession capacity, which we name d_{max}. The highest throughput the network,which we name T_{max}, can only be achieved when all the server receive d_{max} amount of data and the data distributed in each server are fully balanced. The performance of the network balance can be evaluated by the proportion of throughput T in the maximum throughpu T_{max}. As we have:

$$W = \frac{1}{n}\sum\nolimits_{i=1}^{n}[1 - \frac{|d_i - \overline{D}|}{\overline{D}}] \tag{4.10}$$

So we can get:

$$W \geq \frac{1}{n}\sum\nolimits_{i=1}^{n}[1 - \frac{d_{max} - \overline{D}}{\overline{D}}] \tag{4.11}$$

Then this can be derived:

$$\frac{T}{T_{max}} \leq \frac{1}{2 - W} \tag{4.12}$$

And it is similar for the communication balance index in the mesh networks H. In this way, we are able to convert the value of R into network throughput.

These optimizations only solve clustering granularity and DRL model problems mentioned in Section 4.1.3, in order to overcome the obstacles to implementing into the real-world network, the following system model is proposed.

4.3 DRL with LSTM Predictor Model

As DRL consists of the training phase and the decision making phase, and we combine LSTM predictor and DRL model, we introduce the system model of Reinforcement Learning with LSTM predictor in both training phase and decision making phase.

4.3.1 DRL with LSTM Predictor Model at the Training Phase

Unlike in computer simulators, physical IoT networks take time to configure and operate. The physical rules of the world and the cost budget make repeated and unlimited training unfeasible. As a result, building a simulator to mimic the real-world environment can be feasible to provide more data and a better training environment.

To implement the DQN model in a real-world environment, a simulator is to be built to train the DQN model. As the real-world environment has dynamic factors, it is important to make the simulator also dynamic with a similar pattern as the real-world environment. So an LSTM predictor is used to track and learn the key dynamic factor of the real-world environment and generate more synthetic data. Here, in this case, the key dynamic factor is the moving events. We use LSTM predictor to predict the trace of the moving events. With the prediction trace, the simulator can mimic the IoT network based on the LSTM predictor. So the LSTM predictor is used to predict the dynamic factors in the real-world environment, while the other part of the simulator is the static factors of the real-world environment. Thus the real-world environment is simulated with both dynamic and static factors.

Fig 4.6 shows the system model, namely DRL with LSTM Predictor Model at training phase. The DRL model still interacts with the simulator to avoid direct interaction with the real-world environment. In the simulator, an LSTM predictor is built to mimic the dynamic behavior of the real-world environment. Together with the static factors in the simulator, the simulator can form a very similar environment to the real-world one. Also, the data will not be restricted to the limited real-world data, the LSTM predictor can generate synthetic data at run time.

The reason why an LSTM algorithm is used is that not only the recent locations and velocities of the moving events are considered, but also its long-term behavior near certain areas. For example, when the truck inside of a warehouse is loading cargo in a pick-up area, a short-term movement does not necessarily indicate that it will continue moving. The chances are that it just

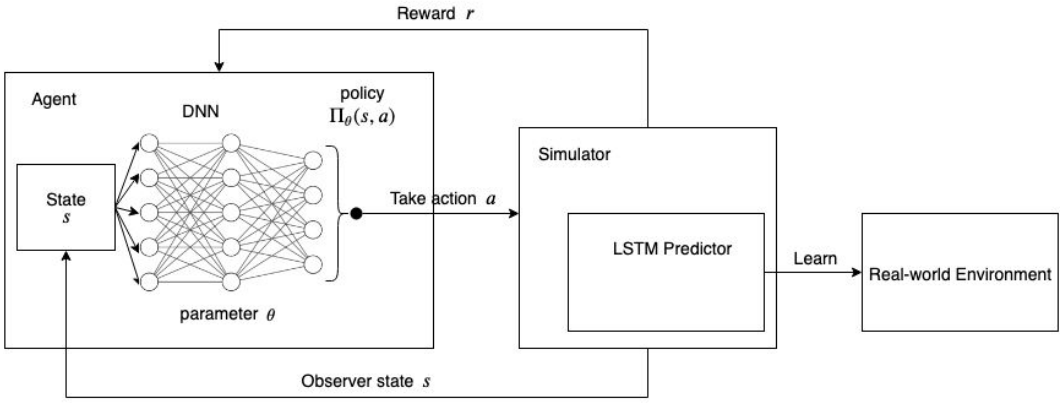


Figure 4.6: DRL with LSTM Predictor Model (at Training Phase)

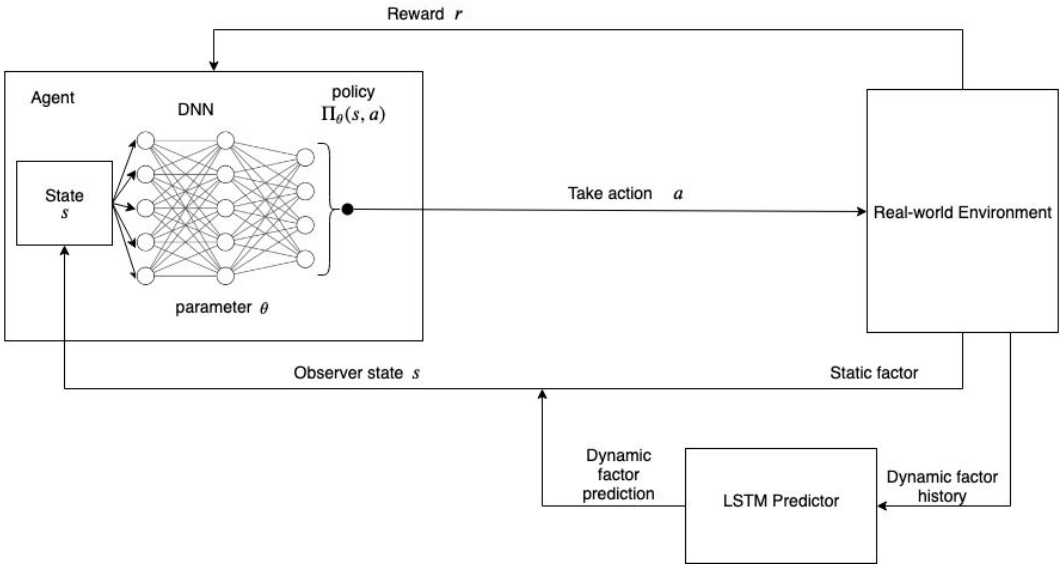


Figure 4.7: DRL with LSTM Predictor Model (Decision Making Phase)

moved a bit to get closer to other cargo within the area, so it just moves a little and will not continue moving in a short time. A simple RNN will not predict this situation well, while an LSTM can have a long-term memory about what happened in certain areas some time ago.

Then the agent interacts with the simulator and learn how to configure the network gradually. The agent observes the environment state (here the environment is the simulation environment), and then uses the DNN and action-choosing policy to choose an action to take. The environment state consists of 4 parts, which are neighbor connection condition, clusters that each node belongs to, event location, and event short-term behavior. The event short-term behavior is the moving direction of the event. Because with the same events position and the same network condition, different event moving direction can influence the correctness of the actions. With the help of event short-term behavior, the DQN can give better solutions for a different moving pattern.

Afterward, DRL model will give the action to the IoT network in the simulator, and the simulator will adjust the network according to the action instruction. After that, the environment will give a reward back to the agent so that the agent can evaluate the effectiveness of the action which was just used and update the parameters in the DNN. In this way, after a large number of explorations and exploitations in the simulator, the agent will develop a suitable model to keep giving right actions in certain states in order to have a higher reward, thus the network in the simulator can maintain a balanced clustering condition.

4.3.2 DRL with LSTM Predictor Model at the Decision Making Phase

The LSTM predictor is not only used at the training phase to generate more training data, but also used at the decision making phase to give DQN model some dynamic prediction about the coming environment state, as shown in Figure 4.7. As DQN uses experience replay in order to eliminate adjacent data correlation, so DRL is not so good at extracting sequential data information. That is why we use the LSTM model to provide dynamic factor prediction for DRL model at the decision making phase.

Here the inputs for the agent are neighbor connection condition, clusters that each node belongs to, event location, and event short-term behavior. The former three can be directly observed from the environment. While the last one, event short-term behavior, needs to be predicted with LSTM predictor. With compound information with both dynamic factor prediction and static factor, the DRL agent can give predictive action output to the real-world environment. The reward from

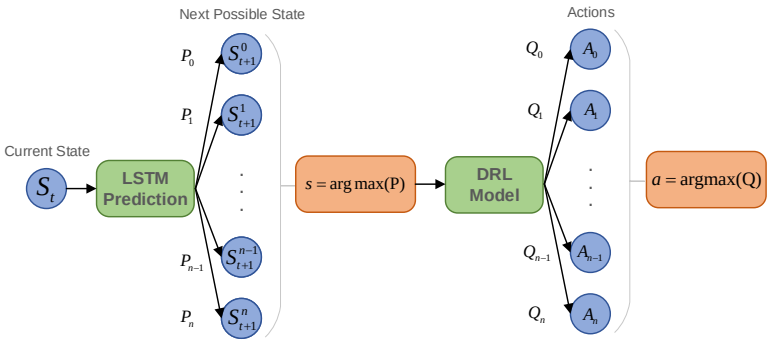


Figure 4.8: Argmax action selection.

the real-world environment is optional. Using real-world environment reward can future improve the performance of the DQN with transfer learning and fine-tuning theoretically, but this is not within the scope of this paper.

In this paper, the main focus is on how to make the LSTM predictor and DQN cooperate with each other. Three connection methods are proposed as follows:

- **Argmax Action Selection**

  This is the simplest and most intuitive method. As shown in Figure 4.8 as LSTM predictor outputs predictions and the corresponding possibilities, the most possible state is the state with the argument of maxima of all the possibilities. Then the state is passed to the DQN. In the same way, DQN will output every action and its Q-value, then the action with the most reward expectation is the action with the argument of maxima of all the Q-values. In the end, this action is used as the output of the DQN model.

  This solution uses argmax function twice, the first of which only reserves the most possible action while the second of which reserves the biggest Q-value. This method is just to fully trust the two DL network and fully believe the output with the argument of maxima of all the possibilities is the correct output. However, all other outputs from the LSTM predictor and DQN are left out.

- **Semi-combined Probability Action Selection**

  This method is an upgrade of Argmax Action Selection. With this method, we only fully trust the output of DQN. All the predictions of LSTM predictor for dynamic object prediction will be used and put into the DQN. The DQN will select the best action for each possible state. The possibility of each action is determined by the LSTM prediction and the same with the state possibility. However, chances are that different states have the same action as output from DQN, so the possibilities are combined for this action. Figure 4.9 shows an example when different state inputs for the DRL Model can generate the same action output. Finally, the action selected for the real-world environment is the action with the highest combined possibility.

  This method reserves the possibilities from the LSTM predictor apart from the biggest one and merges the states with same action output. But still, rewards apart from the biggest rewards in the DQN output are overlooked. So the prediction from the two models is still not fully used.

- **Fully-combined Probability Action Selection**

  With this method, firstly, the LSTM predictor predicts next states and their possibilities. Then each state is passed to the DQN, and the corresponding actions and their Q-values are outputted, which is under the possibility of the state. So the expectation of its Q-value should be the production of the Q-value and the state possibility. The total expectation of

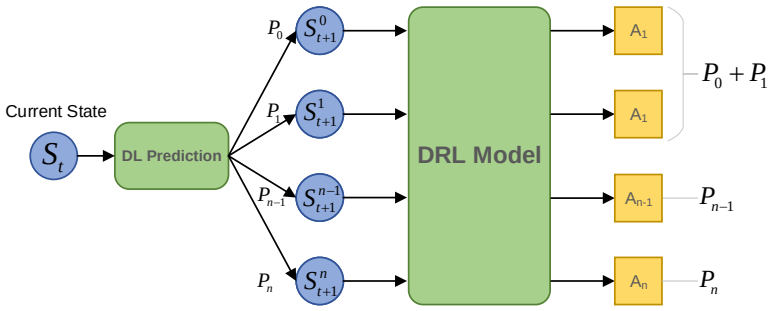


Figure 4.9: An example of Semi-combined Probability Action Selection.

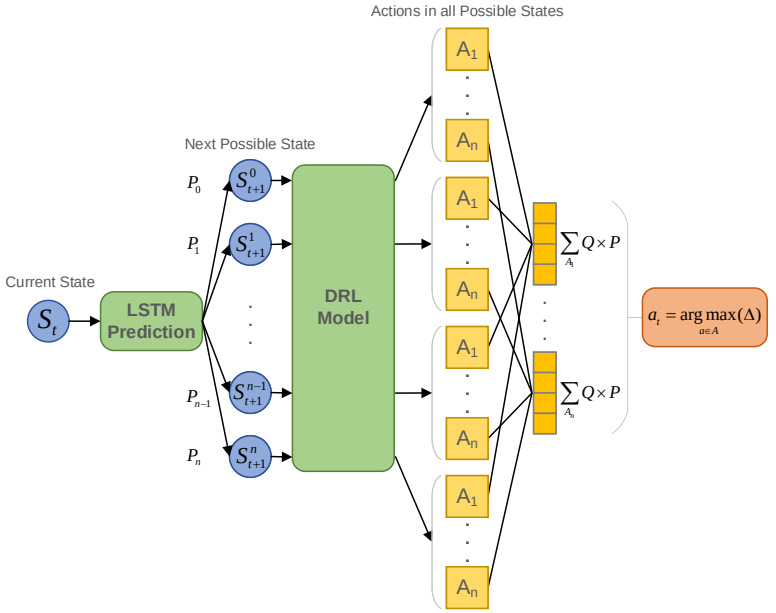


Figure 4.10: Fully-combined Probability Action Selection.

each action is the sum of all these productions in all possible states. Finally, the action with the highest Q-value expectation sum is chosen:

$$a = argmax(\sum_{A} Q \times P) \tag{4.13}$$

In this way, all the possibilities from DL Prediction model and Q-values from DRL model are considered, which avoid the neglect of all the information except for the largest one using argmax function. This makes the model always output the action with the most reward expectation and works well in special cases when different states have the same action output.

To explain the difference between these three schemes, a special case example is given in Table 4.1.

In order to make it easy to comprehend, in this instance, we only consider a DQN with 3 possible states and 3 possible actions. The possibilities of the states that LSTM predictor predicts and the rewards of all actions are listed in the table. If we use Argmax Action Selection, state 1 has the biggest chance so only this state is taken into consideration. In this state, action 3 has the biggest Q-value, so using Argmax Action Selection, we take action 3.

Next Possible State	Possibility	Next Action in This State	Q-value
State 1	0.4	Action 1	1
		Action 2	7
		Action 3	8
State 2	0.3	Action 1	6
		Action 2	5
		Action 3	4
State 3	0.3	Action 1	6
		Action 2	4
		Action 3	2

Table 4.1: A special case example shows how different decision-making schemes influence the result of the prediction.

However, at state 2 and 3, the best actions to take are identical, that is Action 1. So if we use Semi-combined Probability Action Selection and combine the possibility of State 2 and State 3, Action 1 is the best option under 0.6 possibility. While Action 3 is the best option only under 0.4 possibility. So in this case, we also consider the possibility of other states in LSTM predictor and the final output of Semi-combined Probability Action Selection is different, which is Action 1.

However, considering all the possible states is not enough. As the ultimate target in this step is to maximize the expectation of reward, all the Q-values also need to be considered. Summing up the production of all the Q-value and their corresponding possibilities, we can calculate the expectation of reward of all the actions using Fully-combined Probability Action Selection. Then the action with the biggest expectation of reward is the one we want. In this case, the expectations of reward for Action 1, Action 2 and Action 3 are respectively 4, 5.5 and 5. So the action with the most reward is Action 2 and choosing Action 2 can give us better reward with the biggest possibility.

Using Fully-combined Probability Action Selection, as we clarify our target, which is to maximize the expectation of reward, we could work out the output formula and then calculate the expectation of reward of each actions. As a consequence, instead of using argmax function and overlooking part of the outputs, all the LSTM predictor and DQN outputs are considered. So Fully-combined Probability Action Selection can achieve a better output in special cases when LSTM predictor or DQN is not fully trust-worthy theoretically.

In this example, it is shown that with consideration to more outputs of both LSTM predictor and DQN model, the action outputs can be totally different with these three methods. But this is just a special case, and most of the time, when the models make accurate predictions or the highest possibility and reward are greatly larger than the other possibilities and rewards respectively, the outputs of the three schemes are actually the same.

In conclusion, Argmax Action Selection method trust the prediction of the LSTM and DQN too much and overlook possibility information. So it does not work well in certain cases. While Semi-combined Probability Action Selection is an intuitive upgraded solution without sound proof. However, Fully-combined Probability Action Selection method is always selecting the action with the biggest expectation of reward. As a consequence, Fully-combined Probability Action Selection method would give the best overall performance in these three methods proposed, although the difference might not be very significant.

Chapter 5

Experiment Results

5.1 Experiment Setup

5.1.1 Software

The experiment is carried out in a simulation environment, while the truck position data is real-world data from a warehouse. The software for simulation used is PyCharm and for DQN and LSTM models, Keras with a TensorFlow backend is used. For data processing, libraries Numpy and Pandas are used.

5.1.2 Research Scenario

The case-study scenario is based on the real-world application from ProDrive Technologies B.V. as shown in Figure 5.1.

In this experiment, to evaluate the performance of the system fast, we utilize a semi-simulation approach. In the system, we build a simulation scenario, in which all the objects, including sensor nodes, vehicle and servers are simulated. Meanwhile, we collect mobility data from a real testing truck of Prodrive Technologies B.V. as shown in Figure 5.1d.

In this scenario, as shown in Figure 5.1a, unmanned cargo transportation vehicles (trucks) move in the warehouse. And the IoT sensor nodes are randomly scattered into the field, with the constrain that the distance between each two should be longer than certain length. With this constrain, it is easy to derive that the more the nodes are in the field, the density of the nodes get closer to uniform. We build this part in simulation.

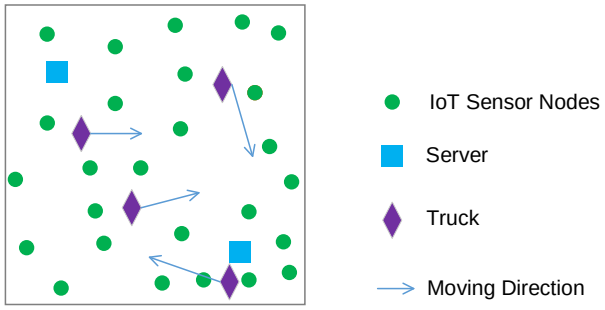
The vehicles are used to move cargo so that the vehicles have a moving pattern. These real mobility data is used in the simulator as the movement trace of the virtual vehicle. So we also use real-world data in our simulation.

We make this semi-simulation for three reasons:

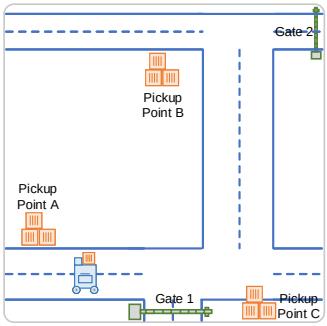
- This makes the system experiment fast.
- We can easily change the IoT parameters in the system.
- We can separate the mobility trace to multiple sub-traces to simulate multiple vehicles because we only utilize one unmanned vehicle in the real area.

In this case-study, IoT sensor devices are deployed in a warehouse to collect sensing data. The IoT devices produce data and transfer these data to the edge servers. The cargo transportation vehicles (trucks) move in the warehouse. The vehicles are used to move cargo with a moving pattern. They trigger the neighbor IoT devices to produce data, e.g. logging data for moving cargo.

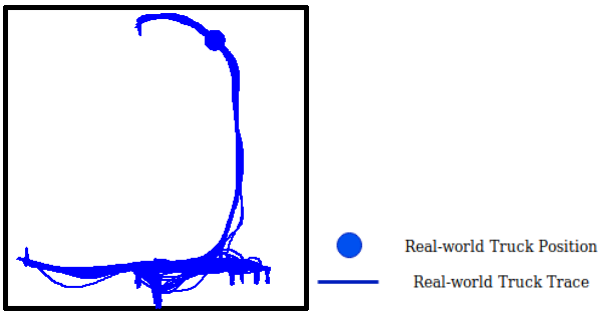
Figure 5.1b shows the layout of the real-world testing area. The vehicle moves in the area to pick cargo at the pickup points, and moves the gates sometimes. In the warehouse boundary,



(a) A scenario example of the IoT network



(b) The map of the warehouse.



(c) Movement trace of the vehicle.

(d) A real testing truck of Prodrive Technologies B.V.

Figure 5.1: Vehicle movement scenario in the warehouse

there are three pickup points and two gates, and the truck runs on the truck pathway and carries out work around or travels between the pickup points. Gate 2 is an emergency gate and normally it is not open so the truck does not go through the gate often. Normally, the work pattern is that the truck enters from Gate 1 and then load or unload around the pickup points.

The direction of real-world moving vehicle has continuous values. To simplify the problem, we discretize the movement trace of vehicle to grid points. We slice the whole testing area into 0.025m×0.025m grid squares. With this method, each location record is inside a square. We use the center of the grid square to represent the real location record. An example of discretizing moving trace is shown in Figure 5.2. After discretizing the moving trace, the LSTM model predicts the dynamic object in 9 prediction behaviors: Up, Down, Left, Right, Up-Right, Down-Right, Up-Left, Down-Left, Stay.

Then we can have an assumption that the truck either stays or moves in a certain length of distance in a time-step. And the whole truck work scope is gridded, so that in each time-step, the truck has 9 behaviors to choose from, and they are staying still or moving to one of the eight directions. The time-step is set small so that the moving trace is still very accurate.

Figure 5.1c shows one sample of the location trace of moving vehicles in the area. In the experiment, we test and evaluate our solution based on this location trace data in this IoT network.

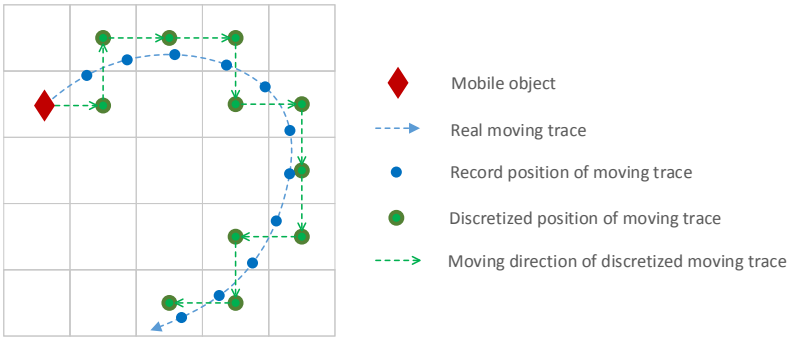


Figure 5.2: Discretize the movement trace of dynamic vehicle.

5.2 Implementing Multiplicative Weighted Voronoi Clustering and D3QN on the DQN-based Dynamic Clustering [1]

5.2.1 Moving Events with Point-to-point Moving Pattern

The DQN model and IoT network parameters are set up according to Table 5.1. Here the experiment uses a simple moving pattern, the dynamic event in the simulator moves from one point to another point recurrently. As shown in Figure 5.3, the Dynamic Clustering Benchmark convergence line is the improved version of the state of the art [1], and it additionally uses Multiplicatively Weighted Voronoi Clustering and D3QN. This will be used as a benchmark for the following experiments. Dynamic Clustering with Target Behavior is simply to add the moving direction of the dynamic target to the original model, which is one of the following 9 directions: stay, up, down, left, right, up-left, down-left, up-right, down-right. The outcomes of the two are similar, they converge to a reward around 0.6. This is because DQN usually uses experience replay to eliminate the dependency of adjacent data. So in nature, DQN does not perform well with temporal sequential data.

The dynamic clustering with multiplicative weighted Voronoi gives better performance. It converges faster and finally stay converged at a higher reward around 0.67. This shows multiplicative weighted Voronoi provides more efficient actions for the IoT network to adjust itself. But the convergence process is still quite slow, which makes the model hard to train. So improvement of the DQN part is needed.

D3QN and Multiplicative Weighted Voronoi Clustering Solution is the best solution among all of these solutions. It not only improves the network granularity for the IoT part but also improves the DQN part by utilizing more powerful DQN algorithms. It converges the fastest with a high convergence reward, which is around 0.72. The improvement in this environment with a simple point-to-point moving patter is about 20%.

5.2.2 Testing D3QN and Multiplicative Weighted Voronoi Clustering Solution with the Real-world Truck Trace

But what if we use the model with the real-world truck moving trace? With limited data from the real-world system, we build a virtual IoT network and the moving event follows these recorded real-world positions. Then we use the D3QN and Multiplicative Weighted Voronoi Clustering Solution to interact with this environment and train on it.

At the training phase, 50% of the whole dataset is used as training data. But not all of these half is always used at the training phase. To test if the solution works well with different amount of training data, we use 25%, 50%, and 100% of these data respectively to test how the amount

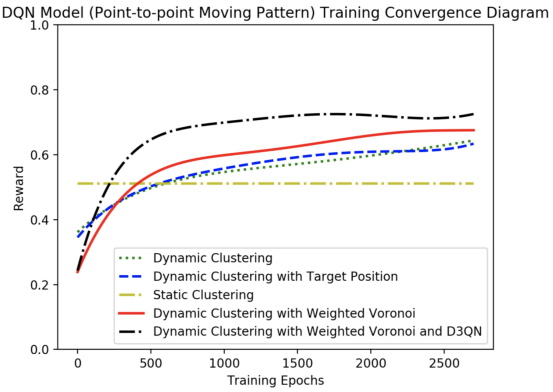


Figure 5.3: Implementing Multiplicative Weighted Voronoi Clustering and D3QN on the DQN-based Dynamic Clustering [1]

Table 5.1: The parameters of experiments.

Parameter	**Value**
Deployment Area	15m×15m
Discretization Square Size	0.025m×0.025m
Cluster-core Move Step Length	1m
Expand and Shrink Rate of MWVC	0.8, 1.25
Node Transmission Range	3m
Normal Packet Size	1Kb
Detection Packet Size	3Kb
Detection Period	10s
Learning Rate	0.001
Discount	0.9
Min Espilon	0.01
No. of DNN Layers	3
Neurons in Layers	600×300×150

of training data would affect the DQN model.

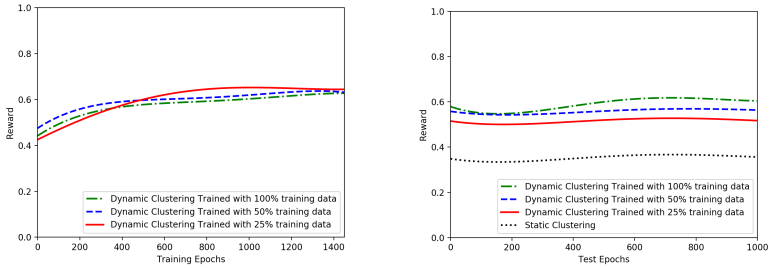
As shown in Figure 5.4a, the convergence speed is similar to that with simple point-to-point moving pattern. At around 500 epochs, the models converge and stay stable. The final convergence rewards of the models trained with different volumes of data set are similar and around 0.63. While the convergence speeds of them are slightly different. The model trained with less training data tend to converge more quickly but at the same time is more prone to problems such as overfitting. This can be tested with cross-validation.

For cross-validation, the remaining 50% of the dataset is used as the test set. And we make the moving event in the IoT network environment follow the trace in the test set. As shown in Figure 5.4b, with the training data getting less, the test performance is becoming worse and approaching the static clustering performance gradually.

The average rewards received from the environment of the models trained with 50% data and 100% data are similar to the convergence result, in which the model trained with 50% data performs worse than the model trained with 100% data. While the model trained with 25% data performs considerably worse with the test data set than the training data set.

This difference shows when the training data set becoming less, the performance of the model is becoming worse. This is because of the following two reasons:

1. As the training data set becoming less, the overfitting problem becomes more severe. Even



(a) Convergence Result (b) Test Result

Figure 5.4: Training and Testing D3QN and Multiplicative Weighted Voronoi Clustering Solution with the Real-world Truck Trace

though L1, L2 regularization, dropout, and early stopping are used, the test average reward is still far lower than the convergence reward by around 20%.

2. In the test data set, there are data that the model is not familiar with. As the training data set becoming less, there will be places that the truck does not cover or cover but with a limited amount of data in the training data. The model performance will significantly decrease when the truck moves around these places in the test set.

So for a limited dataset, it is very important to expand the dataset to tackle overfitting. This is the reason why synthetic data is needed to tackle the overfitting problem caused by limited training data.

Moreover, DQN is a model-based algorithm and uses Experience Replay. The motivation to use Experience Replay is to tackle correlated data and non-stationary distribution problems. As each input and output pair of the sample from Q-learning are correlated to the previous and upcoming input and output pair, these correlation increases the variance of the updated Q-value. Experience Replay eliminate the correlation by randomly select discrete data samples from the memory. However, for our use case, this cancelation of correlation is bad for action selection. For example, when the state of the IoT network is identical, the action should be chosen according to the moving direction of the events, which determines where the dense communication and data generation will be. However, with the cancelation of the correlation of continuous data, it is hard for the DQN to trace the moving pattern from the previous network states. So external event position predictor is needed to get the moving direction and put to the DQN as an input so that the DQN can learn the dynamic factor of the environment.

This proves the necessity to use synthetic data to train the DQN model. So the following experiments are carried out to use LSTM to generate synthetic data at the training phase. To give more direction on the dynamic factors for the DQN model, the LSTM model will also be used at the decision making phase to help the DQN make decisions.

And in this specific case, the key factor is the event positions. As discussed, dynamic clustering with multiplicatively weighted Voronoi and D3QN has better performance than original dynamic clustering, so we use this model as a base model in the following experiments, and we will refer to it as Dynamic Clustering method.

5.3 LSTM Key-factor Prediction Model

5.3.1 LSTM Model

The truck does not necessarily need to come back to Gate 1 every time it loads something, as it could also be organizing cargo inside of the warehouse. As a consequence, it can also go from one pickup point to another one. Moreover, it can just be organizing things within one pickup zone,

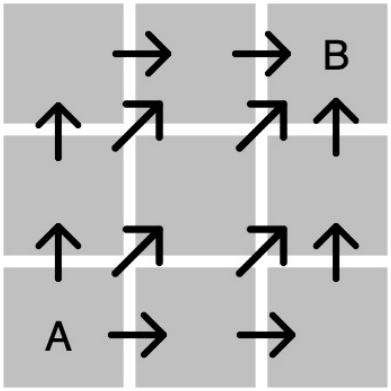


Figure 5.5: Explanation on why LSTM does not have good accuracy

so it could stay loading for some time and then go to somewhere in the pickup zone and unload. These behaviors are not deterministic so the prediction is based on possibilities. The prediction is based on history positions.

The input of the LSTM model is 60 positions of the truck in 60 previous time-steps, in which each position consists of 2 values for x coordinate and y coordinate (say x coordinate parallels to Gate 1 while y coordinate is vertical to Gate 1). The output of the LSTM model is the prediction behavior of the truck in the next time-step. The training data is 50% of all the data log. And the remaining will be used as test set.

Finally, the LSTM model trained with 25%, 50%, and 100% training data achieves 0.692, 0.748 and 0.813 accuracy respectively in the test set. With training dataset getting less, the accuracy of the LSTM model can achieve is decreasing as well. Also, the accuracies the LSTM model can achieve are not very high because, to arrive at certain places, as illustrated in Figure 5.5, there are multiple routes for the truck. In this example, when the truck is going from cell A to cell B, actually at cell A the possibilities of going up, right or upright could be similar because in training set there can be different ways to get cell B in similar situations. In this case, a wrong prediction does not influence the overall long-term direction prediction of the truck, as long as the prediction is in a similar direction. Also bearing in mind that the LSTM model is used for training, actually such deviation can increase the robustness of the DQN model and also decrease overfitting by providing more synthetic data.

In the decision making phase, we use the possibilities of the output to provide more flexibility. For example, the output of LSTM is going up by 90% chance and going upright by 10% chance. Then 90% of times the LSTM will let the truck go up while 10% of times the truck will be directed to upright. Thus all the possibilities are used and the simulator can provide more cases, namely more synthetic data. We use 60 positions of the truck in 60 previous time-steps as the beginning of the trace (which is the blue part in Figure 5.6), and with these data, the next point is the prediction position based on the 60 previous positions. Then we get rid of the oldest position data and append the prediction position as an input to the 59 previous positions, the new 60 previous positions are the input for the next prediction position so that we have a new prediction data. Then we repeat these on and on. With this method, only the first 60 position data are real-world data, and all the remaining are the synthetic data (which is the orange part in Figure 5.6). The first 60 position data can be taken from any time, so the LSTM can start from a random time-step. Here we give a sample of LSTM-based synthetic data.

In Figure 5.6, as shown in Figure 5.6a the blue line shows the real-world truck trace, the truck was going from pickup point C to pickup point A. Then the data synthesis (the red line) starts. The LSTM predictor is able to guide the truck going on the trackway and to pickup point A and back. Then the synthetic data choose to go to pickup point A as Figure 5.6b shows. Only the initial part of the data for these is real-world data, then all the rest is synthetic. Thus from very limited data, the LSTM model can continuously generate unlimited amount of synthetic data.

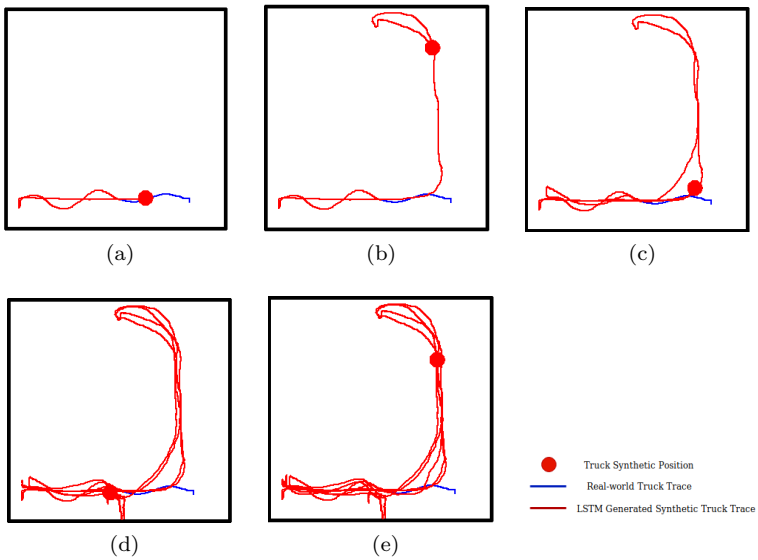


Figure 5.6: An example of LSTM trace prediction.

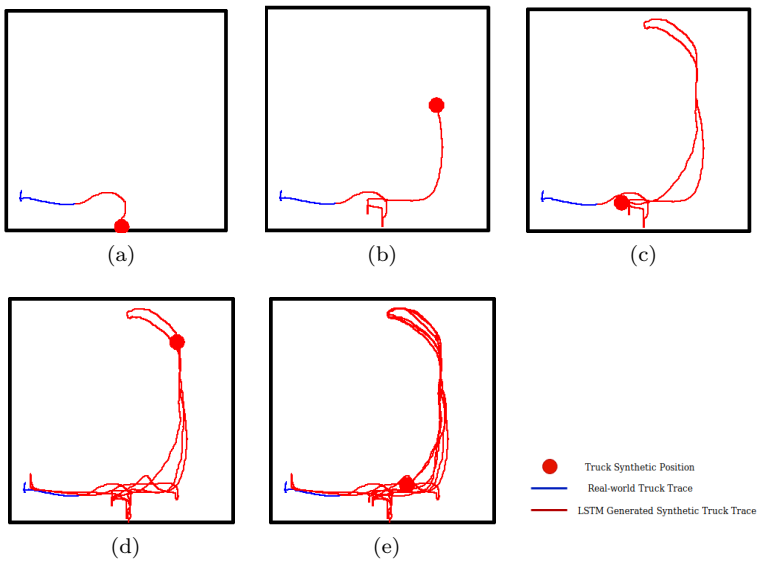


Figure 5.7: Another example of LSTM trace prediction.

In another example, as shown in Figure 5.7, the starting real-world truck position data starts from pickup point A. Subsequently, it is heading to the direction of pickup point C then the synthetic data choose to go to Gate A. In Figuree 5.7b, the synthetic data is going to pickup point B on the trackway from Gate A. It can follow the traceway and mimic the moving pattern of a real-world truck.

Same goes for an abnormal test data sample that starts from around the emergency gate, Gate 2, as shown in figure 5.8. Although the truck enters from an abnormal place as the real-world data (blue line) shown in Figure 5.8a, the LSTM predictor can still predict the trace back to normal working pattern. As shown in Figure 5.8b, the prediction is still able to generate synthetic data in which the truck is loading/unloading cargo in Gate A. Then as shown in Figure 5.8c and Figure 5.8d, the prediction is able to guide the truck in normal work pattern to go to pickup point

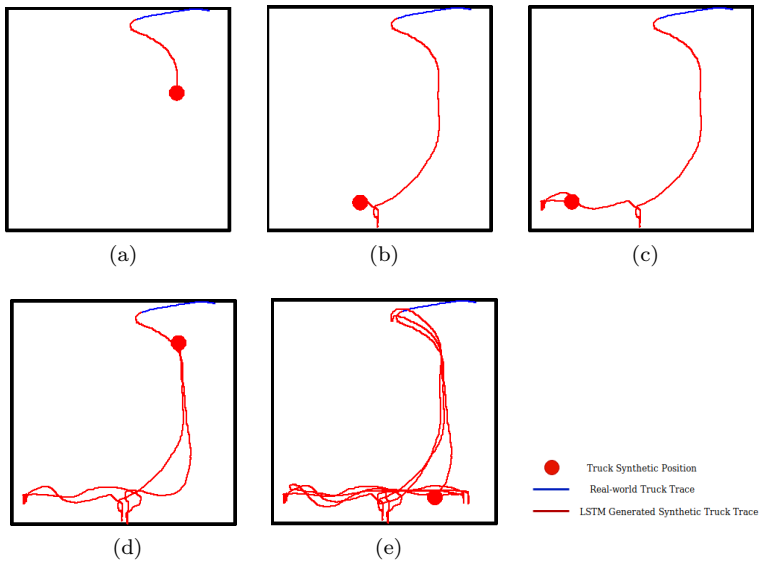


Figure 5.8: An abnormal example of LSTM trace prediction.

A then pickup point B, thus it generates normal synthetic data. This is how LSTM predictor generates synthetic data from a little slice of real-world data.

5.4 DRL with LSTM Predictor Model Experiment

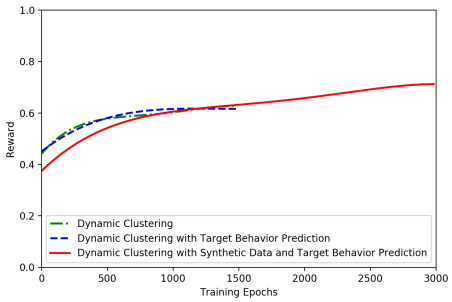
5.4.1 DRL with LSTM Predictor Model (at the Training Phase)

The parameters of the IoT network and DQN model are also set as shown in Tabel 5.1. For each epoch, the Model-based Reinforcement Learning with LSTM Predictor model chooses a random slice of training data set, of which the length is 60 positions. Then with these real-world data, the LSTM predictor starts to predict the afterward data. The environment model will use these synthetic data as the moving event position so that all the training data for DQN is synthetic and unlimited.

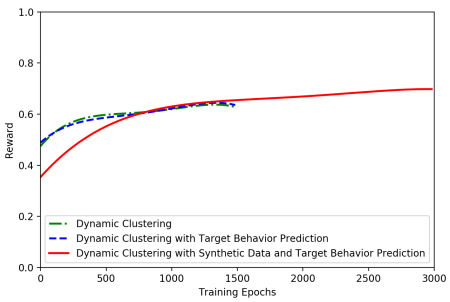
Figure 5.9 shows the convergence results of 4 clusters models with different size of training data. As shown in Table 5.2, Dynamic Clustering is the benchmark DRL model, which does not take the behavior of the moving target as part of the input. While Dynamic Clustering with Target Behavior Prediction takes the behavior of the moving target as part of the input, and in the decision making phase, the LSTM model output will be the behavior prediction input for the DQN model. They show a similar pattern in the convergence process. They both only use real-world data and the final convergence reward of all these three training results stay around 0.6. At around 1500 epochs, they become stable and have converged, which is faster than the

Clustering Method Name	**Trained with**	**Input**
Dynamic Clustering with Synthetic Data and Target Behavior Prediction	Synthetic data	Network State + Dynamic Target Behavior Prediction
Dynamic Clustering with Target Behavior Prediction	Real-world data	Network State + Dynamic Target Behavior Prediction
Dynamic Clustering	Real-world data	Network State
Static Clustering	-	-

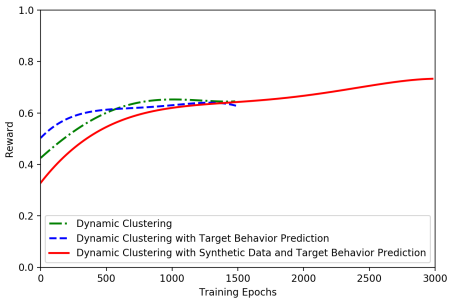
Table 5.2: Different clustering models



(a) 4 cluster and 80 node model convergence result with 100% training data



(b) 4 cluster and 80 node model convergence result with 50% training data



(c) 4 cluster and 80 node model convergence result with 25% training data

Figure 5.9: Convergence results with different size of training data set.

Dynamic Clustering with Synthetic Data and Target Behavior Prediction because the latter has unlimited data in the training data set. Although Dynamic Clustering with Synthetic Data and Target Behavior Prediction and Target Behavior Prediction solution converges more slowly, after around 3000 epochs, they can converge and achieve average reward around 0.7 for all these 3 experiments.

With the data getting less, the two models trained with real-world data become easier to converge, while there is no big difference for Dynamic Clustering with Synthetic Data and Target Behavior Prediction solution. Thus we achieve better system performance by introducing LSTM-based synthetic data. And the improvement in the training phase is around 17%.

Correspondingly, LSTM models for 100%, 50% and 25% training dataset DQN experiments also use 100%, 50% and 25% dataset as training datasets. As mentioned in Section5.3.1, these models respectively achieve 0.813, 0.748 and 0.692 of accuracy on the test set. The three LSTM models will be used correspondingly in the test experiments in Section 5.4.2.

5.4.2 DRL with LSTM Predictor Model (at the Decision Making Phase)

As illustrated in Section 4.3.2, three methods are implemented to the models in the decision making phase. Three experiments are carried out respectively on the same test set with Dynamic Clustering with Synthetic Data and Target Behavior Prediction model trained with 100% training data, and the result is shown in Figure 5.10.

They all achieve slightly less than 0.7 reward on average similar to convergence reward at the training phase. Although the performances are quite similar, as expected, Fully-combined Probability Action Selection still outperforms other solutions as this solution strives to achieving the highest reward expectation. And Fully-combined Probability Action Selection method achieves

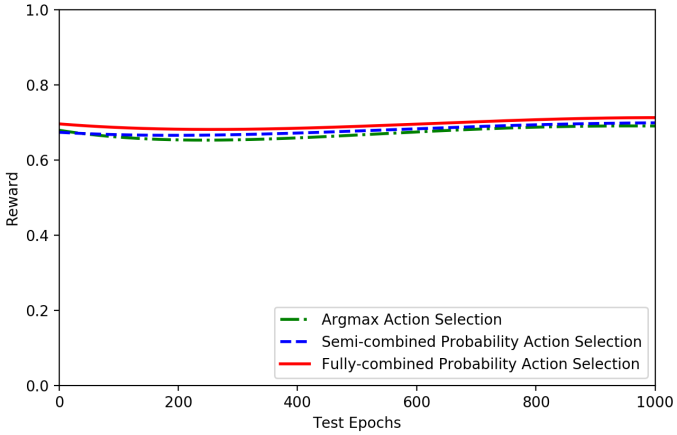


Figure 5.10: Using probability-based models to connect LSTM predictor and DRL model.

around 0.69 reward on average. So in the following experiments, Fully-combined Probability Action Selection will be used in both Dynamic Clustering with Synthetic Data and Target Behavior Prediction and Dynamic Clustering with Target Position and in these two, LSTM will be used to produce target behavior prediction as part of the input for the models. Dynamic Clustering Benchmark does not have the behavior prediction of the moving target as an input so it will not use Fully-combined Probability Action Selection method, and it will act as a benchmark for the other two models.

As shown in Figure 5.11, Dynamic Clustering with Synthetic Data and Target Behavior Prediction, which uses both LSTM generated synthetic data and LSTM behavior prediction as an input gets the best performance in the test dataset. Also, the performance does not drop much as the training data set gets less. The average award is around 0.68 for all these experiments with different size training dataset.

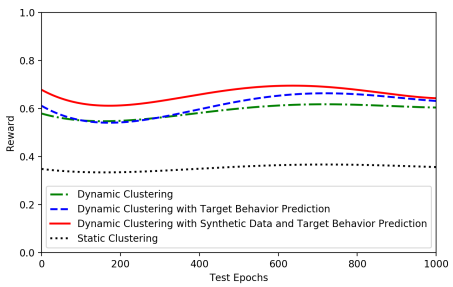
Dynamic Clustering with Target Behavior Prediction outperforms Dynamic Clustering, which shows that DRL is good at static information processing instead of consequent data input. Extracting consequent data input and converting them as a static input can help DRL model make better decisions.

Dynamic Clustering with Target Behavior Prediction has worse performance than Dynamic Clustering with Synthetic Data and Target Behavior Prediction and is prone to the size of training dataset. With 100%, 50% and 25% training dataset, this method achieves respectively 0.62, 0.60 and 0.57 average reward.

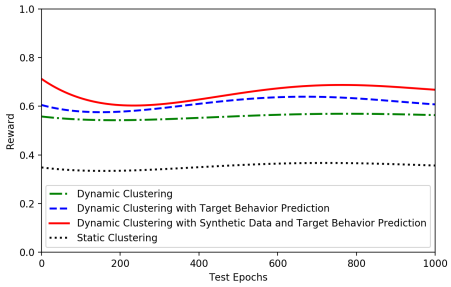
Dynamic Clustering provides the worst performance in these three dynamic clustering methods, and it is very prone to the size of training dataset. With 100%, 50% and 25% training dataset, this method achieves respectively 0.59, 0.57 and 0.51 average reward.

Nonetheless, all of these dynamic clustering methods greatly outperform static clustering, which can only an average 0.12 award.

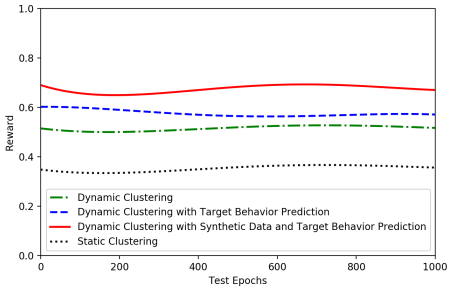
In these experiments, it shows that the method Dynamic Clustering with Synthetic Data and Target Behavior Prediction we propose can improve the system performance by 15% to 30% based on the size of the training dataset. With less data, the method we propose can achieve more improvement on the original model. And it can be predicted that with even less data, the method Dynamic Clustering with Synthetic Data and Target Behavior Prediction we propose will drop slightly but will greatly outperform the model without synthetic data and target behavior predicition.



(a) 4 cluster and 80 node model test result with 100% training data



(b) 4 cluster and 80 node model test result with 50% training data



(c) 4 cluster and 80 node model test result with 25% training data

Figure 5.11: 4 cluster model test results with different size of training data set.

5.4.3 Additional Experiments

To prove the algorithm works with different IoT network setup, additional experiments are carried out with different IoT network parameters.

IoT Networks with Different Amount of Nodes

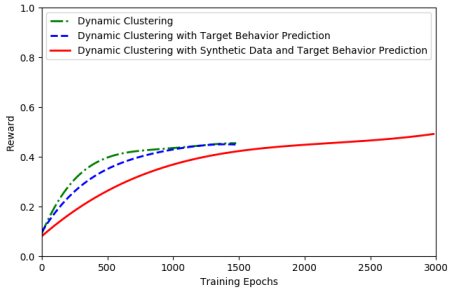
In this set of experiment, all the models are trained with 25% of training data set and use 4 clusters. And the total amounts of IoT network are set to 20, 40, and 80 respectively. The rest parameters of the experiments are the same with the previous experiments, which are shown in Table 5.1.

As shown in Figure 5.12, the pattern of the models with different amount of nodes is similar to the previous experiment. Dynamic clustering with Synthetic Data and Target Behavior Prediction converges the most slowly, but is able to achieve the highest convergence reward. It has a better performance with the test dataset than the other two and it almost achieves the reward it gets at the end of the training phase. Because we constrain the minimum distance between every two nodes, with the nodes get more, the density of the nodes in the field gets closer to uniform. So it is easier to make the IoT networks balanced. That is why a reward increment can be seen as the nodes get more.

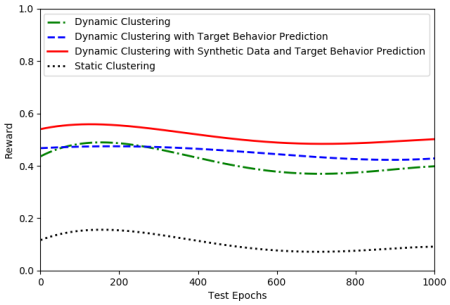
IoT Networks with Different Cluster Numbers

In this set of experiment, all the models are trained with 25% of training data set and have 80 nodes. And the server numbers are set to 2, 3, and 4 respectively. The rest parameters of the experiments are the same with the previous experiments, which are shown in Table 5.1.

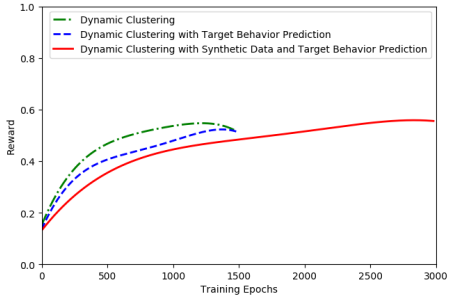
As shown in Figure 5.13, the pattern of the models with different amount of nodes is similar to the previous experiments. Dynamic clustering with Synthetic Data and Target Behavior Prediction converges the most slowly, but is able to achieve the highest convergence reward. It has a better performance with the test dataset than the other two and it almost achieves the reward it gets at the end of the training phase. With the server get less, the number of the average nodes in a server gets more. Because we constrain the minimum distance between every two nodes, then the density of the nodes in the field gets closer to uniform. As a result, it is easier to make the IoT networks balanced. That is why a reward increment can be seen as the cluster number gets fewer.



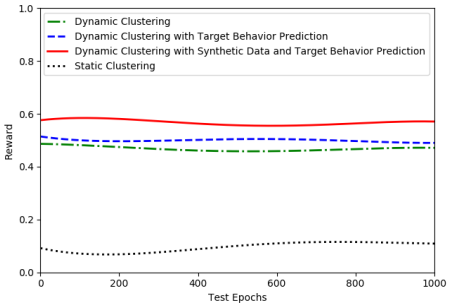
(a) 4 cluster and 20 node model convergence result



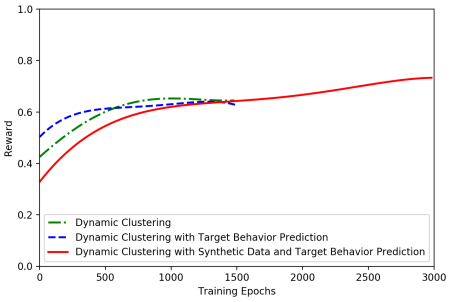
(b) 4 cluster and 20 node model test result



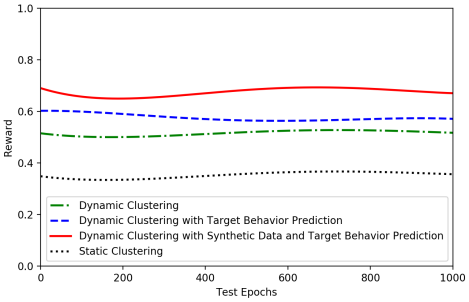
(c) 4 cluster and 40 node model convergence result



(d) 4 cluster and 40 node model test result

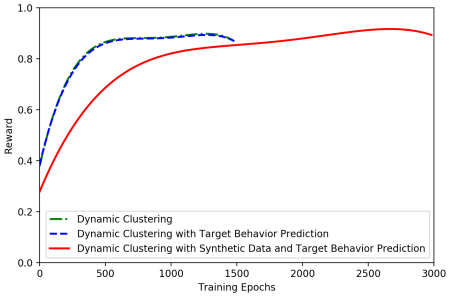


(e) 4 cluster and 80 node model convergence result

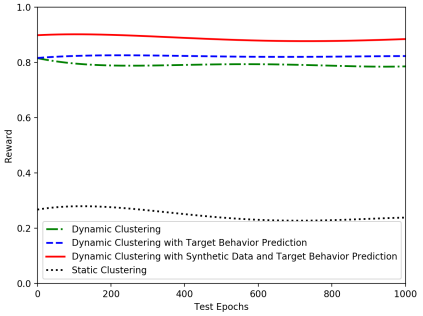


(f) 4 cluster and 80 node model test result

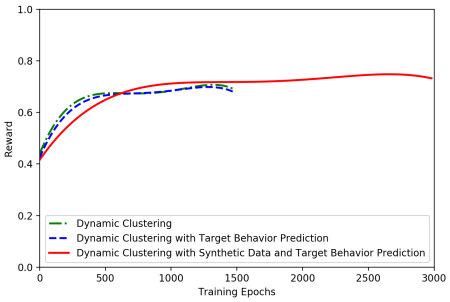
Figure 5.12: 4 cluster model experiment results with different amount of nodes.



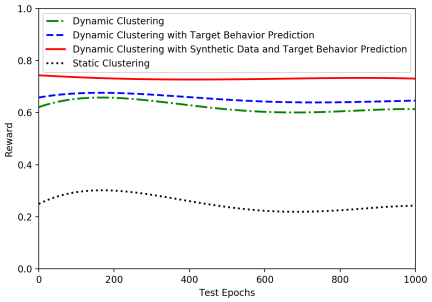
(a) 2 cluster and 80 node model convergence result



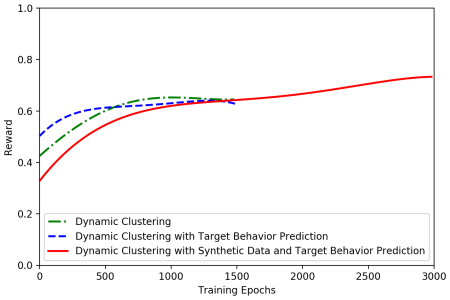
(b) 2 cluster and 80 node model test result



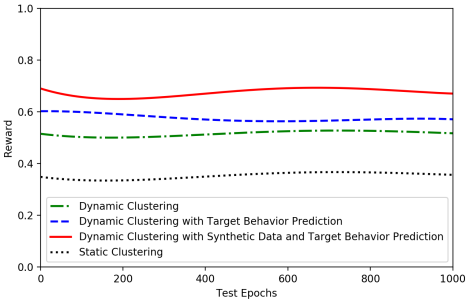
(c) 3 cluster and 80 node model convergence result



(d) 3 cluster and 80 node model test result



(e) 4 cluster and 80 node model convergence result



(f) 4 cluster and 80 node model test result

Figure 5.13: 4 cluster model test results with different number of clusters.

5.5 Experiment Conclusions

In conclusion, first, we reproduce the state of the art and improve it from both the IoT clustering part and the DRL part. In the IoT part, multiplicatively weighted Voronoi diagram clustering method is implemented to provide better clustering granularity and flexibility, while in the DRL part, D3QN is used to provide a more powerful DQN model. The improved model achieves a great outcome with a simple point-to-point moving pattern in simulation with around 20% reward increment.

However, when using limited real-world moving trace data to train the improved DRL model, the model suffers from overfitting problem and performs badly in the decision making phase, especially with small size of training dataset. The bad performance is mainly because of the lack of data.

To tackle this problem, we build an LSTM predictor to mimic the moving pattern of the real-world truck in the warehouse. The LSTM predictor can provide similar a moving pattern and thus generate synthetic data, which makes the position dataset unlimited for the DQN model to train upon.

We use the LSTM predictor for DRL model in both the training phase and the decision making phase. And we call this method Dynamic Clustering with Synthetic Data and Target Behavior Prediction. In the training phase, the LSTM predictor produce synthetic data. While in the decision making phase, the LSTM predictor make behavior prediction.

Furthermore, we propose a solution in the decision making phase to use LSTM to assist the DRL model to consider the dynamic factor in the environment. We propose three methods to make the LSTM model and DQN model cooperate with each other. In the end, the experiment shows that Fully-combined Probability Action Selection we proposed has the best performance.

Dynamic Clustering with Synthetic Data and Target Behavior Prediction can improve system performance by 15% to 30% in terms of reward we defined based on the size of the training dataset compared to the improved DRL model. With less data, the method we propose can achieve more improvement. Compared to static clustering, this method can improve the system performance (reward) by 100% to 400%. Similar results are shown in experiments with different amount of servers and end nodes.

It shows that in the DRL when there is very limited training data, the improvement of using synthetic data is significant. To separate the dynamic factor out from the IoT network and then make it an input to the DQN with experience replay in the training phase, then use another predictor to generate the dynamic prediction as an input in the decision making phase can help the DQN understand the dynamic factor in the environment and have better performance.

Chapter 6

Conclusion and Future Work

In this paper, we design a DRL model for a dynamic IoT network clustering scheme and propose a Dynamic Clustering with Synthetic Data and Target Behavior Prediction method to minimize the error between the simulation and the real-world system.

First, we propose to use Multiplicatively Weighted Voronoi Clustering method as the clustering method because of its good clustering granularity and simple actions to operate with. D3QN is used to improve the DRL performance in terms of efficiency and effectiveness. Then we propose a Model-based DRL Model with LSTM Predictor to port the DRL model from the simulation to the real-world system, and figure out how to minimize the error between the two, namely:

- Use LSTM predictor generate synthetic data in the training phase;
- Use dynamic factor prediction to help DRL model in the decision making phase.

Then we prove that this solution improves the state-of-the-art solution by 15% to 30% based on the size of the training dataset. The synthetic data makes the model less prone to the change of the training dataset size.

For future work, it is possible to port the DRL model into a real-world environment with fine-tuning and transfer learning. A more specific implementation method should be designed, considering more practical details such as how often the server should broadcast the configuration message to the end nodes, if there is any limit and how much is the overhead cost for the IoT devices to switch networks, etc.

Also, other dynamic factors (such as the noise in the environment, user activities, etc.) can be considered and used. Moreover, DRL clustering method can not only meet the requirement of geographical clustering but also be used in logical clustering. For example, similar models could be designed to switch devices between cellular and Wi-Fi, and in this case, both cost and latency could be considered for the reward of the DRL model. The actions could be if a device should be switched to the other network.

Bibliography

[1] Qingzhi Liu, Long Cheng, Tanir Ozcelebi, John Murphy, and Johan Lukkien. Deep reinforcement learning for iot network dynamic clustering in edge computing. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 600–603. IEEE, 2019. iviv, iviv, iviv, iviv, vv, vivi, 18, 19, 20, 22, 24, 33, 34

[2] Sayon Dutta. *Reinforcement Learning with TensorFlow : A Beginner's Guide to Designing Self-Learning Systems with TensorFlow and OpenAI Gym*. Packt Publishing, Limited, Birmingham, 2018. vivi, vivi, 8, 13

[3] Jianfeng Zhang, Yan Zhu, Xiaoping Zhang, Ming Ye, and Jinzhong Yang. Developing a long short-term memory (lstm) based model for predicting water table depth in agricultural areas. *Journal of Hydrology*, 561:918–929, 2018. vivi, vivi, vivi, vivi, 10, 11, 12

[4] *Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence*. Studies in Computational Intelligence, 158. 1st ed. 2008.. edition, 2008. vivi, 19, 21

[5] Meysam Masoudi, Amin Azari, Emre Altug Yavuz, and Cicek Cavdar. Grant-free radio access iot networks: Scalability analysis in coexistence scenarios. 2017. 1

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017. 1

[7] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. 2017. 1, 2, 15, 24

[8] Mehdi Mohammadi, Ala Al-Fuqaha, Mohsen Guizani, and Jun-Seok Oh. Semisupervised deep reinforcement learning in support of iot and smart city services. *IEEE Internet of Things Journal*, 5(2):624–635, 2018. 2

[9] *Internet of Things*. Taylor Francis, 2017. 5

[10] Dimitrios Serpanos. *Internet-of-Things (IoT) Systems Architectures, Algorithms, Methodologies*. 1st ed. 2018.. edition, 2018. 5

[11] Sudip Misra. *Network routing : fundamentals, applications and emerging technologies*. 2008. 5

[12] Mustapha Boushaba, Abdelhakim Hafid, and Michel Gendreau. Node stability-based routing in wireless mesh networks. *Journal of Network and Computer Applications*, 93:1–12, 2017. 6

[13] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 2019. 6

[14] Jie Cao. *Edge Computing: A Primer*. SpringerBriefs in Computer Science. 1st ed. 2018.. edition, 2018. 6

[15] Mahadev Satyanarayanan. Edge computing. *Computer*, 50(10):36–38, 2017. 6

[16] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate iot edge computing with lightweight virtualization. *IEEE Network*, 32(1):102–111, 2018. 6

[17] *Artificial Intelligence in IoT*. Transactions on Computational Science and Computational Intelligence. 2019. 7

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17:lt;xocs:firstpage xmlns:xocs=quot;quot;/gt;, 2016. 7

[19] Habib F. Rashvand and Jose M. Alcaraz Calero. Smart home, smart office. In *Distributed Sensor Systems*, pages 189–219. John Wiley Sons, Ltd, Chichester, UK, 2012. 7

[20] *Machine Learning*. Morgan Kaufmann, 2018. 7

[21] Donna L. Hudson and Maurice E. Cohen. Supervised learning. In *Neural Networks and Artificial Intelligence for Biomedical Engineering*, pages 59–77. John Wiley Sons, Inc., Hoboken, NJ, USA, 2012. 8

[22] *Natural Computing for Unsupervised Learning*. Unsupervised and Semi-Supervised Learning. 1st ed. 2019.. edition, 2019. 8

[23] Csaba Szepesvri. *Algorithms for reinforcement learning*. Synthesis digital library of engineering and computer science. Morgan Claypool, San Rafael, Calif. (1537 Fourth Street, San Rafael, CA 94901 USA), 2010. 8

[24] Anthony L Caterini. *Deep Neural Networks in a Mathematical Framework*. SpringerBriefs in Computer Science. 1st ed. 2018.. edition, 2018. 8

[25] Taesic Lee, Juwon Kim, Young Uh, and Hyunju Lee. Deep neural network for estimating low density lipoprotein cholesterol. *Clinica Chimica Acta*, 489:35–40, 2019. 9

[26] *Recurrent Neural Networks*. InTech. 9

[27] Gm Zhu, L Zhang, Py Shen, and J Song. Multimodal gesture recognition using 3-d convolution and convolutional lstm. *Ieee Access*, 5:4517–4524, 2017. 10

[28] Asma Naseer and Kashif Zafar. Meta features-based scale invariant ocr decision making using lstm-rnn. *Computational and Mathematical Organization Theory*, 25(2):165–183. 10

[29] Christopher Watkins and Peter Dayan. Q -learning. *Machine Learning*, 8(3-4):279–292, 1992. 12

[30] Makoto Katoh, Ryota Shimotani, and Kiyoshi Tokushige. Integrated multiagent course search to goal by epsilon-greedy learning strategy: Dual-probability approximation searching. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 388–392. IEEE, 2015. 13

[31] Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in dqn. 2018. 15

[32] E.A.O. Diallo and T. Sugawara. Learning strategic group formation for coordinated behavior in adversarial multi-agent with double dqn. volume 11224, pages 458–466. Springer Verlag, 2018. 15

[33] Ying Huang, Guoliang Wei, and Yongxiong Wang. V-d d3qn: the variant of double deep q-learning network with dueling architecture. In *2018 37th Chinese Control Conference (CCC)*, volume 2018-, pages 9130–9135. Technical Committee on Control Theory, Chinese Association of Automation, 2018. 15

[34] Harald Sundmaeker, Patrick Guillemin, and Peter Friess. *Vision and challenges for realising the Internet of things.* Publications Office, Luxembourg, 2010. 17

[35] Shusen Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27, 2017. 17

[36] Qiang Fan and Nirwan Ansari. Towards traffic load balancing in drone-assisted communications for iot. *IEEE Internet of Things Journal*, 6(2):3633–3640, 2019. 18

[37] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. 2018. 18

[38] Imad Alawe, Adlen Ksentini, Yassine Hadjadj-Aoul, and Philippe Bertin. Improving traffic forecasting for 5g core network scalability: A machine learning approach. *IEEE Network*, 32(6):42–49, 2018. 18

[39] Chaoyun Zhang and Paul Patras. Long-term mobile traffic forecasting using deep spatio-temporal neural networks. 2017. 18

[40] Laisen Nie, Xiaojie Wang, Liangtian Wan, Shui Yu, Houbing Song, and Dingde Jiang. Network traffic prediction based on deep belief network and spatiotemporal compressive sensing in wireless mesh backbone networks. *Wireless Communications and Mobile Computing*, 2018, 2018. 18

[41] Nan Jiang, Yansha Deng, Osvaldo Simeone, and Arumugam Nallanathan. Cooperative deep reinforcement learning for multiple-group nb-iot networks optimization. 2018. 18

[42] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717. IEEE, 2017. 18

[43] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017. 18

[44] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. 2017. 18

[45] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. 2016. 18

[46] H. Zou, Y. Zhou, J. Yang, H. Jiang, L. Xie, and C.J. Spanos. Deepsense: Device-free human activity recognition via autoencoder long-term recurrent convolutional network. volume 2018-, page lt;xocs:firstpage xmlns:xocs=quot;quot;/gt;. Institute of Electrical and Electronics Engineers Inc., 2018. 18

[47] Xiaojiang Tang, Li Tan, Anbar Hussain, and Minji Wang. Three-dimensional voronoi diagrambased self-deployment algorithm in iot sensor networks. *Annals of Telecommunications*, 74(7):517–529. 19

[48] Lauro C. Galvo, Antonio G.N. Novaes, J.E. Souza de Cursi, and Joo C. Souza. A multiplicatively-weighted voronoi diagram approach to logistics districting. *Computers and Operations Research*, 33(1):93–114, 2006. 24