Research on Network Fault Detection and Diagnosis Based on Deep Q Learning

Peipei Zhang(✉), Mingxiao Wu, and Xiaorong Zhu

Nanjing University of Posts and Telecommunications, Nanjing, China
1342113203@qq.com

Abstract. In order to improve the efficiency and quality of service of the network, network convergence and the development of heterogeneous network have became inevitable. It is a challenge to detect and diagnose the various network faults efficiently in the complex network environment. To solve this problem, a network fault detection and diagnosis algorithm based on deep Q-learning is proposed. Combining deep learning and reinforcement learning model to classify network faults, we can classify some obvious network states via using less features, and filter irrelevant or redundant features at the same time. Results show that the algorithm can use less features to achieve higher classification accuracy, and the accuracy can reach 96.7%.

Keywords: Heterogeneous wireless networks · Deep Q learning · Fault diagnosis · Fault prediction

1 Introduction

With the development of 5G and 6G, we can forecast that in order to meet the needs of users, the network environment will be complex in the future. Under the trend that network develops more and more heterogeneous and intensive, how to diagnose and predict network faults effectively has become a huge challenge, which has been studied by many experts. Szilagyi [1] proposed a complete fault diagnosis framework. The fault detection process mainly monitors the radio measurement data and compares it with the normal behavior captured by the profile, and the diagnosis of the root cause depends on the historical fault cases, and we should figure out these cases' impact on the performance indicators. Three key performance indicators (KPI) are considered in the research process in that literature, namely, channel quality, call drop and early handover time. Khanafer [2] used the simulation data and the actual data to verify, but the identification of the faulty cells only depends on one KPI, that is, the call drop rate. Khatib [3] proposed a diagnosis method based on supervised genetic fuzzy algorithm. Using genetic algorithm to learn fuzzy rule base depends on labeled training set. The experiment is based on a simulation data set and a real data set with 72 records, four kinds of KPIs and four fault causes are considered in the process of fault diagnosis. From the experts' studies on network fault diagnosis, the traditional network fault diagnosis algorithms are based on supervised

Q. Wu et al. (Eds.): WiSATS 2020, LNICST 358, pp. 533–545, 2021.
https://doi.org/10.1007/978-3-030-69072-4_43

learning and rely on a huge number of data sets. What's more, these algorithms only consider a few kinds of faults and the process of fault identification only relies on a small number of KPIs. However, in the complex heterogeneous wireless network environment, network faults will become more diversified, and the identification of network faults will also rely on more KPIs. Therefore, we propose a network fault detection and diagnosis algorithm based on deep reinforcement learning, which is used to solve the situation that the network state is diverse.

Reinforcement learning (RL) [4] is an important branch of machine learning. The essence of reinforcement learning is to describe and solve the problem that the agent learns strategies to maximize rewards or achieve specific goals in the process of interaction with the environment. Unlike supervised learning, reinforcement learning does not tell the agent how to take correct actions. It only evaluates the actions and corrects the action selection and strategy according to the feedback signals. Therefore, the return function of reinforcement learning needs less information and is easier to design, which is suitable for solving more complex decision problems. Recently, with the rise of deep learning (DL) [5] technology and its brilliant achievements in many fields, deep reinforcement learning (DRL) [6], which integrates deep neural network and RL, has become a research hot spot of all parties, and has made great breakthroughs in computer vision, robot control, large real-time strategic games and other fields.

The main contribution of this paper is combining deep learning and reinforcement learning model to classify network faults, specifically, classifying some obvious network states via using less features, and filtering irrelevant or redundant features. Simulation results show that the proposed algorithm can achieve accurate network fault diagnosis and prediction.

2 System Model

2.1 Network Scenario

Figure 1 shows a heterogeneous wireless network scenario in which macrocell, microcell and femtocell overlap each other. In this scenario, due to the diversity of the network, the system becomes more complex and the network management becomes more difficult. We take the network fault diagnosis and prediction in this scenario as an example to verify the value of the network fault detection and diagnosis method based on deep reinforcement learning.

2.2 Network Fault Data Set

The network fault data set comes from the simulation environment set up, which is generated by OPNET 18.6. In this simulation environment, the building of cellular network and the setting of base station parameters are shown in Table 1.

In the simulation, 11 kinds of network status categories are mainly set, which can be divided into five categories: normal, interference, coverage, hardware and transmission. Among them, normal is $\{FC_1\}$, interference is divided into two types: uplink and downlink interference $\{FC_2, FC_3\}$, coverage fault $\{FC_4\}$, hardware is divided into four

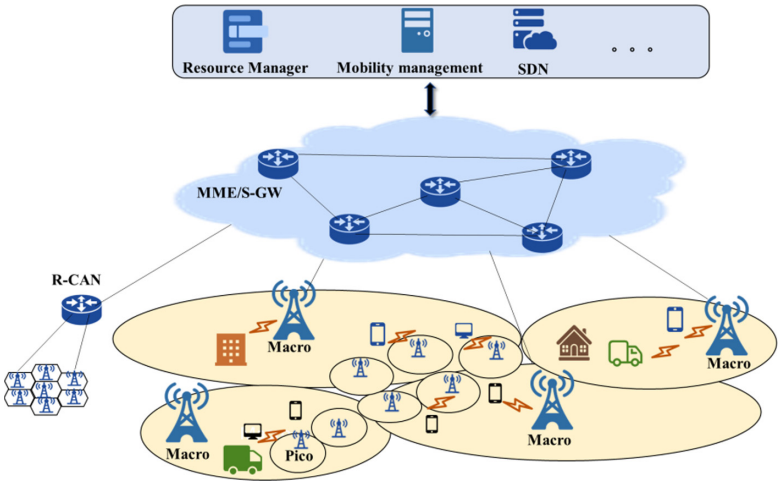

Fig. 1. Heterogeneous wireless network scenario

Table 1. Network simulation parameters

Simulation parameters	Marcocell	Mircocell
Number of base stations	3	8
Number of users/base station	30	10
Transmission power/dBm	46	30
Standard deviation of shadow fading/db	8	10
Propagation loss model	Free Space	Indoor office environment
Antenna gain/dBi	15	8
Operation mode	LTE,5 MHz,FDD	LTE 10 MHz FDD
Receiving sensitivity/dBm	−110	−107
Base station selection strategy	Best suitable eNodeB	Best suitable eNodeB
User distribution	Random distribution	Random distribution

different base station faults $\{FC_5, FC_6, FC_7, FC_8\}$, and transmission is divided into three different link failures $\{FC_9, FC_{10}, FC_{11}\}$. That is, $C = \{FC_1, FC_2, FC_3, \ldots, FC_{11}\}$.

For each network state, 16 kinds of key performance indicators are used to measure in the simulation, and the specific indicators are shown in Table 2. Then, the occurrence time of these network states is set in advance in order to generate data labels manually. Each simulation time is set to 2 h, and the occurrence time of each network state is 20 min. In the end, about 10000 pieces of data were obtained.

Table 2. Network KPI parameters

Symbols	Description
RSRP	Reference Signal Receiving Power
RSRQ	Reference Signal Receiving Quality
PD_UL	Uplink Packet Loss Rate
PD_DL	Downlink Packet Loss Rate
SNR_UL	Uplink SNR
SNR_DL	Downlink SNR
RRC	Radio Resource Control connection establishment success rate
E-RAB	Evolved Radio Access Bearer connection establishment success rate
DCR	Drop Call Rate
HO	Handover Success Ratio
CUAT	Cell Uplink Average Throughput
CDAT	Cell Downlink Average Throughput
LT (→)	Average Link Throughput (→)
LT (←)	Average Link Throughput (←)
HO_d	Handover Delay
LER	Link Error Rate

3 Markov Decision Process of Network Fault Identification

In order to use reinforcement learning algorithm to solve the problem of network fault identification, it is necessary to model the problem of network fault identification as Markov decision process. As shown in Fig. 2, when an agent is carrying out a task, it first interacts with the environment to generate a new state, and the environment returns a reward. As the cycle goes on, the agent and environment constantly interact to generate more new data. Reinforcement learning algorithm [4] is to generate new data through a series of action strategies and environment interaction, and then use the new data to modify their own action strategies. After several iterations, the agent will learn the action strategies needed to complete the correct fault diagnosis.

We define (x, y) as a sample in data set, vector x as the value of feature set $k \in K = \{k_1, k_2 \ldots k_n\}$ composed of key performance indicators, and $y \in Y$ as the target label. c is defined as the cost function. When a new feature k_i is adopted, the cost is $c(k_i)$. This paper defines environment state space S, observation state o, action space A, reward function $r(s, a)$, environment transformation function $t(s, a)$ as follows:

$s = \left(x, y, \tilde{K}\right) \in S$ consists of a set of samples (x, y) and the currently selected feature set $\tilde{K}$. The observation state $o = \{x_i, k_i\}$ is the state accepted by the agent, and it has no target label. Action is $A = \{A_C, A_K\}$, where A_K represents selecting a new feature that has not been selected before, and A_C means to use a classification action to predict

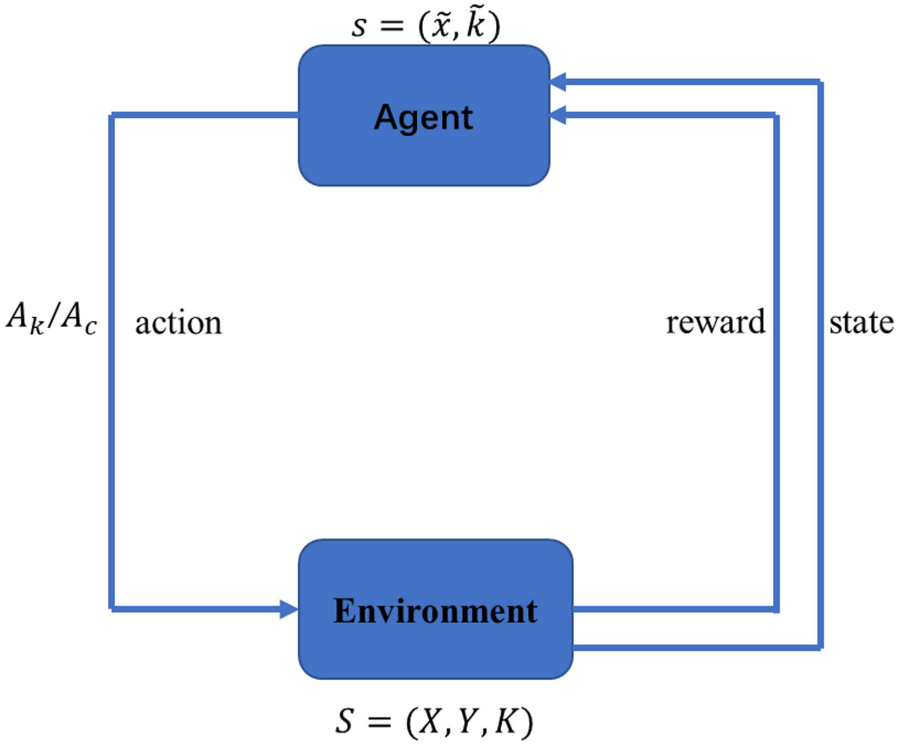

Fig. 2. Markov decision process for network fault identification

which category the sample belongs to. We provide that if $A = A_C$, the event will end (see Fig. 3).

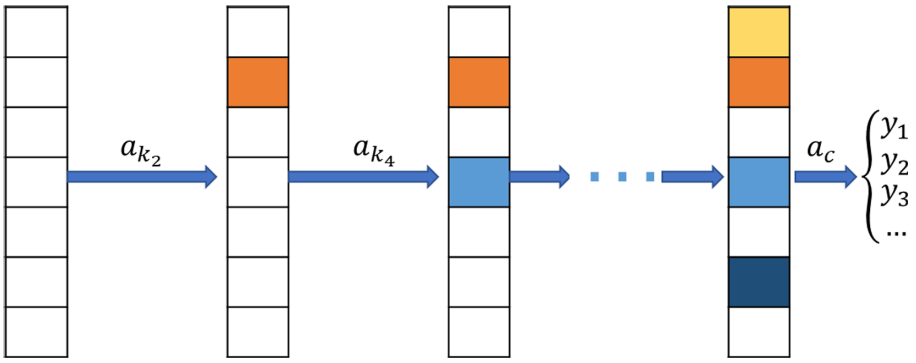

Fig. 3. Classification process

The reward function $r(s, a)$ is a quantitative evaluation performed for each state. Since the purpose of the algorithm is to achieve the correct classification and the selection

of the optimal feature subset, the reward function should be designed to achieve the correct classification and seek the optimal subset features based on this, that is, the reward function should punish the wrong classification action. In addition, the reward function should also consider the problem of finding the optimal subset features. To sum up, in each step, the reward function can be defined as:

$$r((\tilde{x}, y, k), a) \begin{cases} 0 & if\ a \in A_C, a = y \\ -1 & if\ a \in A_C, a \neq y \\ -\mu c(k_i) & if\ a \in A_k, a = k_i \end{cases} \tag{1}$$

When the action of the agent is selected as the classification action, and the selected action is the final correct classification, the reward is defined as 0; when the action of the agent is selected as the classification action, but the selected action is the wrong classification, the reward is defined as -1; in the third case, when the action is selected as a feature selection, the reward is defined as $-\mu c(k_i)$, in fact, we take it as the result of feature selection, and the purpose is to minimize the expected cost. The cost factor μ is a constant, which affects the number of selected features.

The environment transformation function $t(s, a)$ is defined as:

$$t(s, a) = \begin{cases} T & if\ a \in A_C \\ \left(x', y, \tilde{K}\right) & if\ a \in A_k \end{cases} \tag{2}$$

where T represents the termination state and $\left(x', \tilde{K}\right)$ represents the next state when a new feature is selected. In one cycle, the most states are converted to $|K| + 1$. When $a \in A_C$, the environment changes to the termination state. When $a \in A_k$, it enters the next state of selecting new features.

4 Deep Q Learning

The most typical model-free reinforcement learning is Q-learning. In ordinary Q-learning, when the state and action space are discrete and the dimension of them is not high, Q-Table can be used to store the Q value of each state action pair, while when the state and action space are high-dimensional continuous, it is not practical to use Q-Table. In the actual network fault diagnosis process, the network structure is very complex, the corresponding causes of network fault are also very diverse, and the parameters for measuring network performance are also various, so the state and action space studied in this paper is high-dimensional continuous, using q-table is not realistic. The method of this paper is to use DQN [7] to turn the Q-Table updating problem into a function fitting problem, and the similar state can get the similar output action. The formula makes the Q function approximate the optimal Q value by updating the parameters as follows:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{3}$$

We define the state-action value $Q(s, a)$, which represents the state-action value function that takes action a while following the strategy π when the agent is in the state s.

Strategy π refers to the probability distribution of actions under a given state, defines the behavior of the agent in a specific environment at a specific time, and can be regarded as a mapping from environmental state to actions. Here, the strategy determines how the fault recognition task should identify the fault category correctly or select the appropriate feature subset in the current state. $Q^*(s, a)$ represents the optimal state-action value function, it subjects to the Bellman equation [4]. If the optimal value of s' at the next time step is known to all actions a', then the optimal strategy is to choose action a' to maximize expected value:

$$Q^*(s, a) = E_{s' \sim t(s,a)}\left[r + \gamma \max_{a'} Q^*\left(s', a'\right) | s, a \right] \tag{4}$$

γ is the discount factor, and r represents the reward value of the current state taking action.

DQN uses a neural network function with weight θ as an approximator, called a Q network. One Q network can be iteratively trained by minimizing the loss function of the decision sequence:

$$Loss(\theta) = E\left[\left(Q_{target} - Q(s, a; \theta) \right)^2 \right] \tag{5}$$

where $Q_{target} = r + \gamma \max_{a'} Q^*\left(s', a'; \theta'\right)$, and θ' is the parameter of the fixed target network.

The DQN training process is shown in Fig. 4. It uses two key technologies. One is the experience reply. The function of the experience reply is mainly to solve the problems of correlation and non-static distribution. The specific method is to store the transfer samples $\left(s, a, r, s'\right)$ obtained from the interaction between the agent and the environment at each time step in the playback memory unit, and randomly take out a minibatch of them to train when training is needed. This treatment breaks the correlation between samples and makes samples independent of each other. The other one is a fixed target value network (Fixed Q-target): calculating the network target value requires the use of the existing Q value, now a network with slower updating rate is used to provide this Q value. This improves the stability and convergence of training. To make the algorithm performance more stable, establish two neural networks with the same structure: one network that has been always updating neural network parameters (MainNet) and another one network for updating Q values (TargetNet).

Initially, assign MainNet's parameters to TargetNet, then MainNet continues to update the neural network parameters, while the TargetNet parameters are fixed. After a while, the MainNet's parameters are assigned to TargetNet, and so on.

In this way, the target Q value is stable for a period of time, which makes the algorithm update more stable.

5 Algorithm

To facilitate the simulation, we map the observation state o to $(\overline{x}, \mathrm{m})$, the vector $\overline{x}$ is a masked vector of the feature set x composed of key performance indicators, it contains

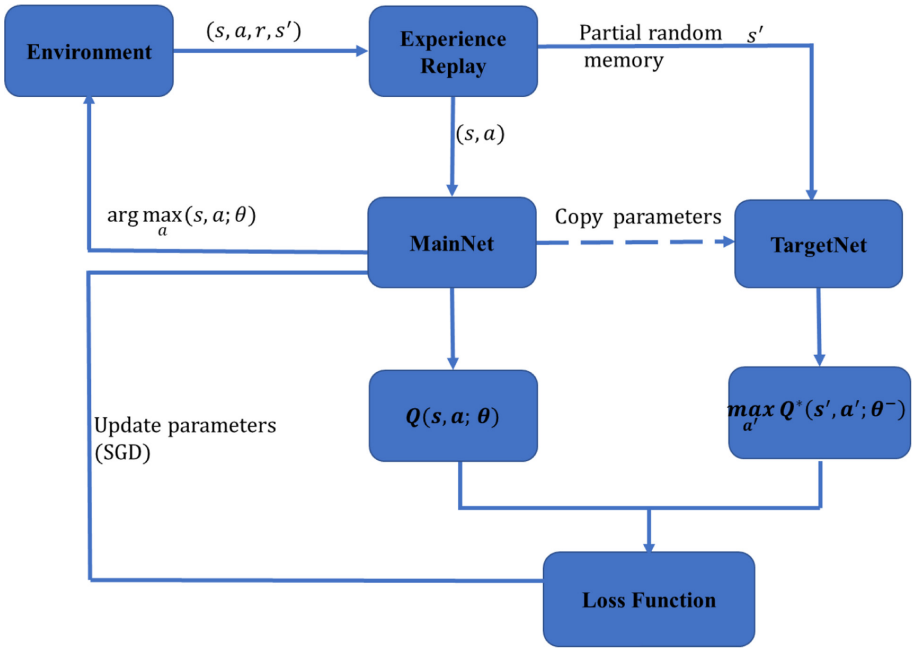

Fig. 4. DQN training process

two parts, one is a optional feature value, another one is 0, indicating that an unknown value is selected.

$$x_i = \begin{cases} x_i \ if \ k_i \in K \\ 0 \ \ otherwise \end{cases} \tag{6}$$

Mask m $\in (0, 1)^n$ is a vector, and different values indicate obtaining different types of features. If the obtained feature value $k_i \in K$, value is 1, otherwise it is 0.

$$m_i = \begin{cases} 1 \ if \ k_i \in K \\ 0 \ otherwise \end{cases} \tag{7}$$

The architecture of the model is shown in Fig. 5. The input layer consists of feature vector $\bar{x}$ and binary mask m, then a neural network, and the final fully connected layer outputs the q value of classification and feature selection actions. Algorithm 1 and Algorithm 2 describe the algorithm and environment simulation.

Fig. 5. DQN model framework

Algorithm 1 Environment simulation

function STEP(s, a)

if $a \in A_k$ **then**

Select a corresponding position feature:k= $k + k_a$

Create the mask $m^{(i)} = 1$ if $k_i \in K$, *otherwise* zeros

Return (k, m), $-\mu c(k_i)$

else if $a \in A_c$ **then**

$$r = \begin{cases} 0 & if\ a = y \\ -1 & if\ a \neq y \end{cases}$$

Draw a new sample from the data-set reset the environment

Return $(done, r)$

end if

end function

Algorithm 2 DQN Training

Randomly initialize networks parameters θ and θ'

Initial the environment E with $s_0 \in (x, y, \varnothing)$

Initial the experience pool P with size N

For episode =1, EPOCHS **do**

for $e \in E$ **do**

repeat

Simulate one step with ε -greedy police π

$a = \pi(s),\ (s', r) = STEP(s, a)$

Store the transition (s, a, r, s') into pool P

until $|P| > N$

Start to train the net and update P circularly, choose a mini batch of experience from P, Set:

$$y^{DQN} = \begin{cases} r \\ r + \gamma \max_{a'} Q(s', a'; \theta') \end{cases}$$

Perform a gradient descent step on:

$$loss(\theta) = (y^{DQN} - Q(s, a; \theta))^2$$

end for

end for

6 Experimental Results

The data obtained by setting the simulation environment is input into the DQN model as input parameters, and the reward is obtained through continuous iteration. The accuracy of network fault classification is shown in Fig. 6 and Fig. 7. As can be seen from Fig. 6, in the initial stage of training, the average reward convergence rate is very fast, and then tends to be stable. Since we narrowed its range to [−1,0], this effectively overcomes the numerical explosion. It can be seen from Fig. 7 that the optimal classification accuracy rate is 96.7%. During the initial training, the DQN algorithm selects almost all features to make classification decisions, which makes the accuracy rate increase rapidly. As the number of iterations increases, the number of selected features will gradually decrease. After a period of exploration, the agent will find the smallest subset composed of important features, and as the number of iterations increases later, the total number of choices changes slowly and tends to be stable. In this process, the agent will continue to change the best subset of feature combinations to improve accuracy, but it has not improved a lot. Therefore, the classification accuracy rate increases rapidly at the beginning, and gradually converges to a stable value after reaching higher accuracy.

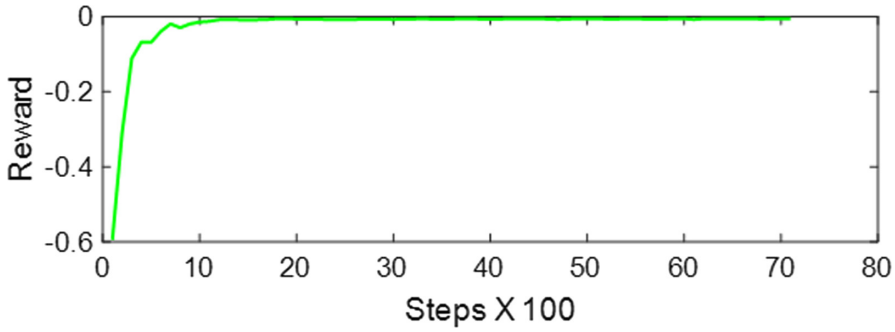

Fig. 6. Changes of reward value under different iterations

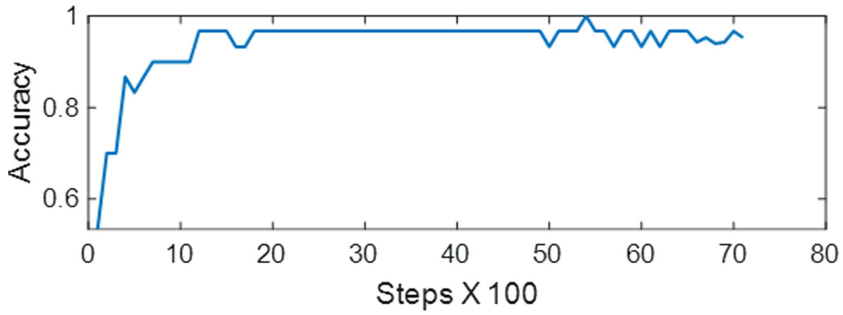

Fig. 7. Changes of fault classification accuracy under different iterations

Figure 8 is a simulation result for selecting different cost factors μ. From the picture, we can see that the smaller μ is, the smaller the absolute value of the cost generated by

feature selection, then the agent will iterate more times to search for the optimal feature subset. At first, the algorithm selects almost all features, then as the number of iterations increases, then it starts filtering redundant features. The speed of filtering redundant features at $\mu = 0.001$ is significantly lower than the rate at $\mu = 0.01$. As the curve stabilizes, the difference in classification accuracy between the two is very small. So we can choose to control the best choice and convergence speed by adjusting to cost factor μ.

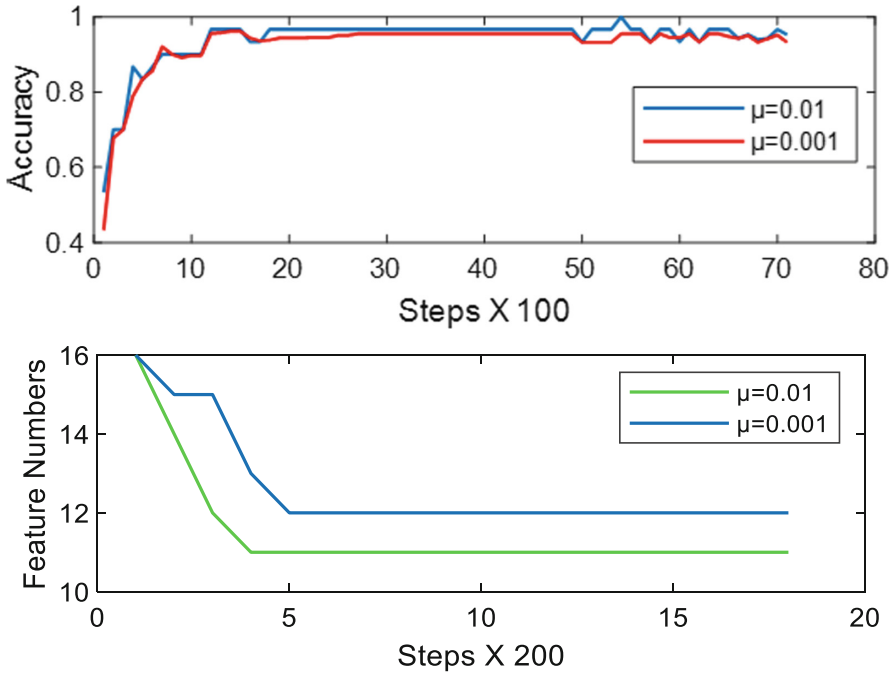

Fig. 8. Changes of fault classification accuracy and number of feature combinations under different cost factors

7 Conclusion

The algorithm proposed is different from the traditional supervised learning algorithm. It combines deep learning and reinforcement learning models to classify network faults, classify some obvious network states via using less features, and filter irrelevant or redundant features. Simulation results show that this method can achieve more accurate network fault diagnosis and prediction. However, it requires a lot of computing resources during the training process, and the training time is slow, so other enhancement technologies can be selected in the future to improve performance.

References

1. Szilagyi, P., Novaczki, S.: An automatic detection and diagnosis framework for mobile communication systems. IEEE Trans. Netw. Serv. Manage. **9**(2), 184–197 (2012)
2. Khanafer, R.M., Solana, B., Triola, J., et al.: Automated diagnosis for UMTS networks using bayesian network approach. IEEE Trans. Veh. Technol. **57**(4), 2451–2461 (2008)
3. Khatib, E.J., Barco, R., Andrades, A.G., et al.: Diagnosis based on genetic fuzzy algorithms for LTE self- healing. IEEE Trans. Veh. Technol. **65**(3), 1 (2015)
4. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)
5. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
6. Henderson P, Islam R, Bachman P, et al.: Deep reinforcement learning that matters. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
7. Janisch, J., Pevn, T., Lis, V.: Classification with Costly Features using Deep Reinforcement Learning (2017)