REINFORCEMENT LEARNING BASED WIRELESS BASE STATION PARAMETER OPTIMIZATION

A Thesis

by

KUN YANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tie Liu
Committee Members,	Krishna Narayanan
	Xiaoning Qian
	Anxiao Jiang
Head of Department,	Miroslav M. Begovic

December 2019

Major Subject: Electrical Engineering

ABSTRACT

Nowadays, in order to provide customers with the best surfing experience through the wireless network. Companies started building modern wireless base stations with a large amount of algorithm-based parameters that can optimize the performance of a single base station. However, tuning these base stations to reach their best performance is not only a time-consuming task but also requires experts for tuning. Trying to make this tuning procedure more efficient, we introduced deep reinforcement learning and built a policy that can optimize a single KPI with a similar performance as the human experts.

In our paper, we claimed to achieve the following accomplishments,

- Built a simulator that can accurately describe a wireless base station's parameter tuning scenario in the real world. The simulator enables the estimation of specific Key Performance Indicator (KPI) while can give rewards as feedback to the tuning actions made by a human or the policy.

- Employed deep reinforcement learning, together with imitation learning and prioritized experience replay, to build an agent that can automatically tune the parameters for the base station with the performance better than human experts.

DEDICATION

To my parents, my girlfriend and all my friends who have already been a support in my hard time.

To my advisor Dr. Tie Liu for his careful guidance during my research time.

ACKNOWLEDGMENTS

First of all, I would like to express my most profound gratefulness to my advisor, professor Tie Liu who helped me with his careful and thorough guidance over my research. Without whom, I will never have an opportunity to finish an exciting program like this.

I would also like to thank my committee members, Dr. Chao Tian, Dr. Krishna Narayanan, Dr. Anxiao Jiang, and Dr. Xiaoning Qian, for their encouragement, valuable comments, and thoughtful questions.

Furthermore, my sincere thanks will also go to Dr. Cong Shen from the University of Virginia for his valuable advice on training the reinforcement learning agent and validating the dataset.

I thank my fellow labmates, Ruida Zhou, Chenjie Luo, and Qiang Zhang, for the discussions over various researching topics, for the practices before the defense, and for the happy time, we spent together inside the same lab.

Last but not least, I would like to thank my parents, Jiancheng Yang and Xiaolei Zhang, and my girlfriend, Yuni Tian, for their spiritual support for all the time.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. Tie Liu, Dr. Chao Tian, Dr. Krishna Narayanan as well as Dr. Xiaoning Qian of the Department of ECEN and Professors Anxiao Jiang of the Department of CSCE.

Great thanks to Dr. Cong Shen from the University Of Virginia for his excellent advice on building the RL agent and validating the data.

Funding Sources

There's no funding source for this project.

NOMENCLATURE

KPI	Key Performance Indicator
RL	Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
NN	Neural Network
BS	Base Station
MDP	Markov Decision Process
OU	Ornstein-Uhlenbeck
TD	Temporal Difference

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

1.1 Problem Introduction

The wireless station system is complex and needs a vast amount of effort to optimize its performance. In traditional problems dealing with wireless base stations(BS), people usually concentrate on how to distribute resources for the whole system. These models have a well understood physical system, and they are more critical for building up an efficient wireless system, as shown in 1.1. By allocating those resources, we can optimize the performance of the whole system. However, optimized parameters at the system level cannot guarantee optimal performance for a single cell, so the companies come up with BSs that have algorithm switches that can adjust a single cell's performance(figure 1.2).

1.1.1 System Description

As shown in figure 1.2, our BS system works under the following strategy. The BS collects full information from the environment, including the environment variables, current BS state, and nearest-neighbor BS state. After collecting these pieces of information, the system is expected to adjust to a better performance automatically each time instance.

However, the reality is that due to the high complexity of the system, the companies are not able to build these autonomous systems yet. They still need tons of experts to manually tune those parameters, which is not only expensive but also time-consuming. Based on this fact, our goal is to build a system that can tune these parameters and get a competitive result with human experts.

1.1.2 Data Set

The dataset has base station cell data collected in different districts from a metropolitan in a period of seven months(May - Nov 2018). The dataset contains 5077 cells and overall of 325,3776 data points, and human experts tune all data points.

Typically, we expect a tuning process ends within 21 days. Nevertheless, there exists special BSs whose KPI is hard to optimize; in these situations, we will record the BS as a 'bad cell' and

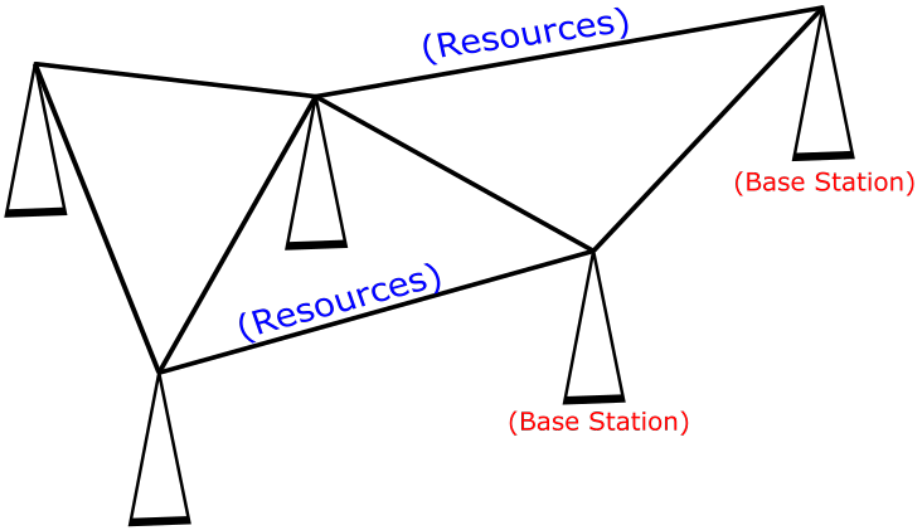


Figure 1.1: Traditional Wireless BS System

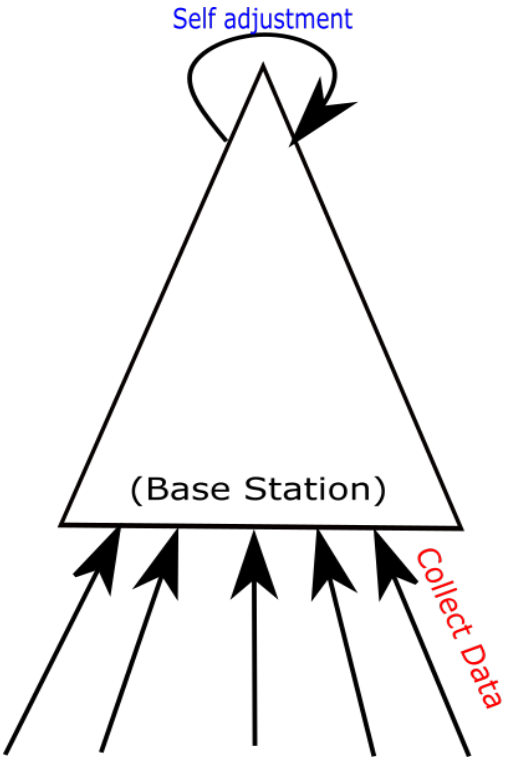


Figure 1.2: Modern Setting For Wireless BS

keep on tuning.

From figure 1.3, we can see that for all the BSs, most of their adjusting procedure lasts less than 26 days. The most common ones are eight and 21 days. However, at the same time, we do observe certain BSs whose tuning process requires more than 200 days.

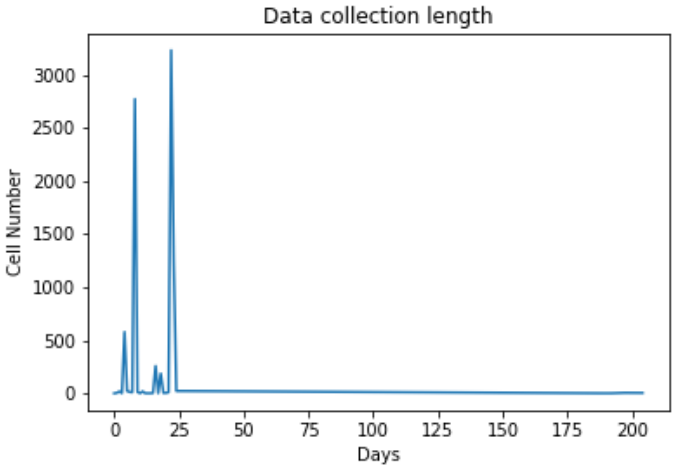


Figure 1.3: The tuning process length distribution of the dataset

1.1.3 Potential Methods

As human experts tuned all the data from our original data set, it is natural to take advantage of imitation learning, i.e., to build a system that purely imitates all the moves from the experts.

Moreover, as the experts are making decisions each time instance based on the current state, we can form the whole system as a Markov Decision Process(MDP), which we describe as the formula1.1.

$$\mathbf{P_A}(S, S') = \mathbf{Pr}(S_{t+1} = S' | S_t = S, A_t = A) \tag{1.1}$$

Here S and S' belong to the state space $\mathbf{S}$; A belongs to the action space $\mathbf{A}$. $\mathbf{P_A}(S, S')$ is the probability that at state S, taking action A will lead to state S'.

Reinforcement learning (RL) has been proved influential in solving MDP problems [1]. So, we regard Deep RL as another potential approach to solve the problem.

Furthermore, considering the potential improvement combining imitation learning and reinforcement learning, we also made an effort to implement these two together in our research. Which we will talk about in detail in chapter 4.1.3.

1.2 Related Works

The traditional wireless network system uses optimization methods with various problem settings for power allocation problems [2, 3, 4, 5, 6] as well as bandwidth distribution problems[7, 8]. The optimization methods are the proved method that can solve problems with a clear functional relationship between the objective parameters and control of the resources.

However, the modern systems are growing too fast; two significant problems come up on the way for traditional optimizing methods,

- The system is too complex for optimization methods to solve.
- The relationship between the control parameters and the objective parameters is not clear.

So more and more people turn to look for help from other methods. Deep RL is one of them. Deep RL becomes famous since Google conquered Atari games in 2015[9] and introduced Alpha-Go that beat human professional player[10] in 2016. From then on, people are using Deep RL to conquer harder games or help to solve control problems that were hard to solve.

Lastly, in recent years, people have found a successful path to implement Deep RL into various wireless communication systems, including BS activation control[11], connected vehicles[12], and BS content caching[13].

2.1 System Scenario

We consider a wireless network system inside a particular sector, and each BS works together with each other under a well-optimized system-level wireless system. Starting from that point, the BSs still need to optimize their performance based on their integrated algorithm switches. So they what to collect data from the environment and make decisions based on the collected data.

The experts can only adjust the parameters once every day, but the system will collect the data every hour. Due to the drastically change (shown in figure 2.1) of the KPI over a single day, even the experts can not make the KPI improve for the whole time, so we use the daily average of our KPI as the metric that decides the whether the system is improved or not.

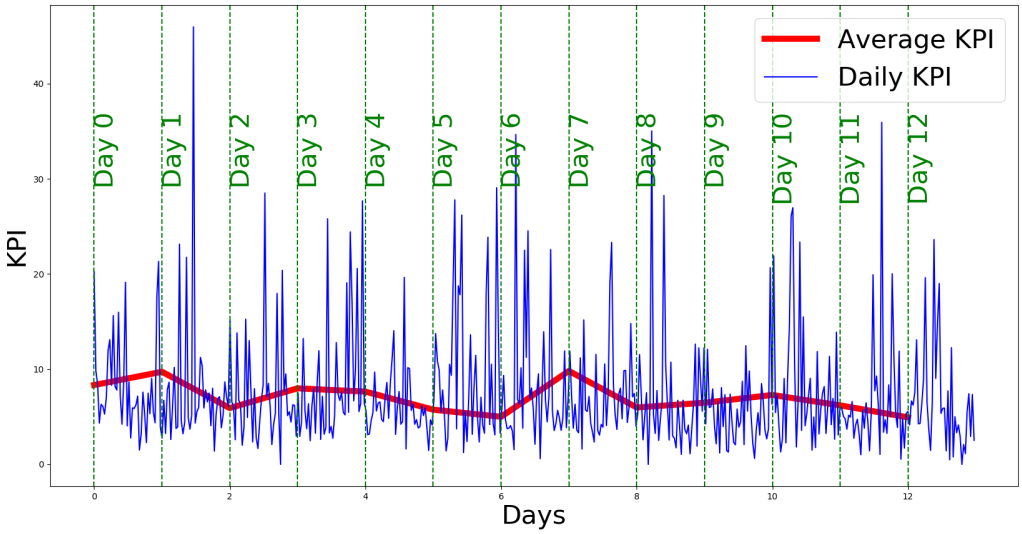


Figure 2.1: KPI change of a typical expert tuned base station

2.2 KPI for optimization

There are multiple KPIs in this system that we can tune, but optimizing them together at the same time is a challenge and may not be a reasonable place to start. Based on this fact, we choose the KPI with a well-defined standard of improvement.

The KPI we chose, at last, is the ratio that a customer's download throughput is less than five megabytes per second. We claim that this KPI is an excellent choice as a starting point for that we know several algorithm switches do have a strong relationship with this KPI. Thus the tuning is sure to be meaningful. We denote that the KPI is a function of the states (state variables) and the actions (parameters we can tune),

$$K_t = f(S_t, A_t) \tag{2.1}$$

Where f is a function mapping the state and the action pair to the KPI, but the exact mapping of this function is complicated and remains unknown. So our thought is to build a Deep Neural Network (NN) to do the regression and thus estimate the function, which we will describe in the next chapter.

2.3 Problem Formulation

We have a clear goal for our system, i.e., decrease our chosen KPI as much as possible. However, before we formulate our system, we would like to make it clear that the system in the real world is noisy. So that even a single step adjustment might have already improved, the KPI could also reflect as a negative response. Furthermore, the KPI itself might also be affected by states that are not predictable nor controllable, e.g., the amount of the customers using the cell or the performance of the nearest neighbor. Last, as a practical problem, the tuning time cannot be infinitely long; if the system cannot get improved in three weeks, we will regard this cell as a bad cell just as we already described in chapter 1.1.2.

Based on these facts, our reward function of the system should contain three parts:

The system should give a reward/punishment for each step we take, but this single step tuning

is not reliable enough because of the noisy environment, so this part should remain small. The second term should reward/punish the total change of the system, which compares the difference between the KPI with the initial KPI at the end of a single tuning procedure (an episode). The third term punishes the agent for no able finding an optimal solution in three weeks (result in a 'bad cell'). The form of the reward function is shown in the following equation.

$$R_t(S_t, A_t) = R_{t,Single-Step} + R_{t,Stuck} + R_{t,Timeout} \tag{2.2}$$

After making the base structure of the reward function clear, we can then form our MDP problem. As we already introduced in section 1.1.3, this problem is an MDP problem, typically an MDP problem can be described as a tuple of five elements $M = (\mathbf{S}, \mathbf{A}, P, R, \gamma)$, where $\mathbf{S}$ is the state space, $\mathbf{A}$ is the action space, P is the probability distribution, R is the reward function for the system and γ is the discount factor determines how far we would like to see into the future.

The system works as the following, the agent observes a state S_t at time t and makes a decision accordingly. After applying the action to the environment, the agent will receive a reward R and observe a new state S_{t+1}. We design our MDP problem setting as following:

- **1).State:** In chapter 1 we divided states based on the system they belong to. Although that setting is direct and easy to understand, it is not helpful for our formulation. Here, we form the state as the following four parts.

$$S_t = (S_{stationary}, S_{t,random}, S_{t,predictable}, A_{t-1}) \tag{2.3}$$

  Where stationary states mean that if the BS is determined, we can never change these state variables again, so this part of the system is not changing. Random state variables mean that this part of the state changes randomly as time varies, i.e., the evolution of these states are not closely related to KPI or actions. Predictable ones are those states have a strong relationship with the KPI so that we can almost surely predict its value after we take a step. Finally, A_{t-1} is the action we took at the previous time instance. The purpose of dividing

the states into these parts will be explained in detail in chapter 3.2.2, the state transfer part.

- **2).Action:** The actions are vectors with each of their dimensions is in continuous value space; we deal with these successive action spaces using the following strategies.
  - Use the deterministic actor networks rather than stochastic ones.
  - Discretize each action to 10 dimensions after the deterministic network, in order to increase the exploration speed.

- **3).Reward Function:** As we stated at the beginning of this section, our reward function should contain three parts, and we define the reward function as Equation 2.4

$$\begin{aligned} R_t(K_t(A_t, S_t)) \quad &= C_1 * \tfrac{(K_{t-1}(S_{t-1},A_{t-1}) - K_t(S_t,A_t))}{K_{t-1}(S_{t-1},A_{t-1})} \\ &+ \mathbb{1}_{Episodic} * C_2 * \tfrac{(K_0 - K_t(S_t,A_t))}{K_0} - \mathbb{1}_{Timeout} * C_3 \end{aligned} \tag{2.4}$$

Here K_t is the KPI function that represents the KPI at time slot t. C_1, C_2, C_3 are three constants and $C_2, C_3 >> C_1$, K_0 represents the initial KPI. $\mathbb{1}$ is the indicator function that indicates the end of an episode and the end of the three weeks. At each time slot, we would like to choose the action that maximizes the long term reward. This accumulated long term reward can be represented through the Bellman Equation 2.5.

$$Q(S_t, A_t) = R(S_t, A_t) + \gamma * \min_{A_{t+1}} \mathbb{E}[Q(S_{t+1}, A_{t+1})] \tag{2.5}$$

So that our optimization problem becomes

$$\begin{aligned} \min_{A_t} \quad & Q(S_t, A_t) \\ s.t. \quad & A_t \in \mathbf{A} \end{aligned} \tag{2.6}$$

So our objective is to find the best action A that solves problem 2.6 at each time instant.

Knowing that RL is powerful in solving MDP problems [14, 15, 16, 17], we adopt the RL method to help us solve this problem, more specifically, we used Deep Deterministic Policy Gradient(DDPG)[18] to build the system and deal with the continuous action space.

Every RL system follows a similar structure as shown in figure 3.1, but our system is facing challenges while building the system, one of our accomplishments is that we built a stable environment dealt with all these challenges.

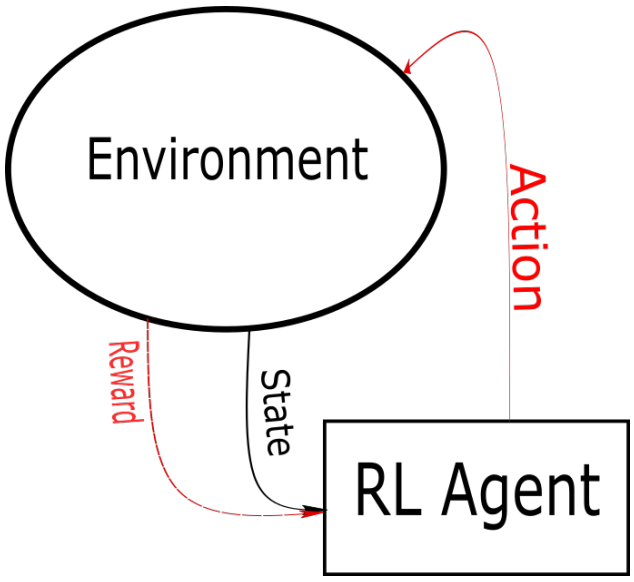


Figure 3.1: A standard RL system

3.1 Challenges

The very first challenge is that we only have a static dataset, which means we don't have access to the real environment; neither do we have a simulator that is close to the real-world scenarios. Our solution is to build a simulator that can accurately describe the environment. The simulator we built contains two parts, the KPI regressor, which is in charge of estimating the KPI function.

We created this regressor using a deep NN, and we will discuss this in section 3.2.1. The other part is the state transfer estimate, and we used a rule-based naive assumption to build the state-to-state transfer system, which we will talk about in detail in section 3.2.2.

Another challenge is that the original dataset contains more than 1300 continuous state and action variables. Dealing with a system with this dimension is hard. Moreover, since we are aiming to build a system to optimize a single KPI, the original dataset will, for sure, have some redundancy. To conquer this challenge, we used network pruning on the KPI regressor to reduce the dimension of the whole system. We will go through the details together with the KPI regressor.

Finally, we come to the challenge that all state and action spaces are continuous. We have mentioned this before and would like to use DDPG to solve this problem, as shown in section 3.3.

So combining this three-part, we eventually build a system that looks like the figure 3.2. The system contains two main parts, the simulator, and the RL agent.

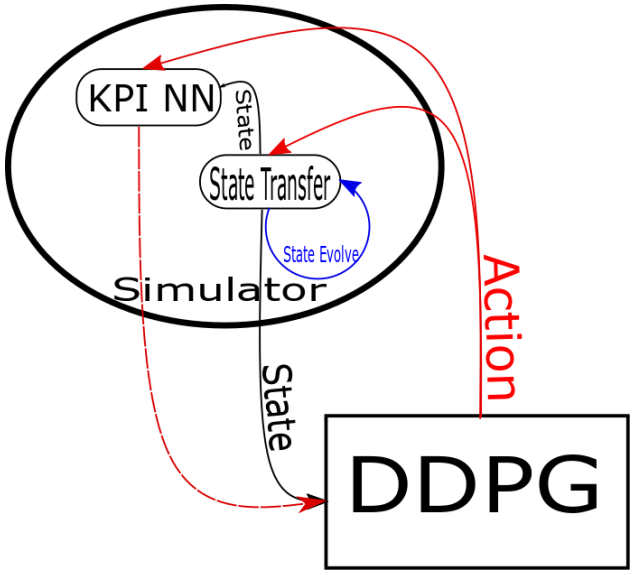


Figure 3.2: System Structure

3.2 Simulator

To build a reliable simulator for training, our simulator contains two main parts, the KPI regressor and the state transfer.

3.2.1 KPI Regressor

As stressed previously, we build the regressor using deep NN. To cut the dimension of the NN, we first build a large NN and train it for limited epochs until its loss (mean absolute error) is reasonably small. Then we apply network pruning to eliminate undesired states and build a new NN with these new states. The network structures all contained in appendix A.1

The original network pruning method will not guarantee the reduce the dimension of the input space; so we make small changes to the algorithm. First, we only work on the very first fully connected layer, so that we can directly see the relationship between the input variables and the first layer. Secondly, instead of throwing all the weights that are smaller than a threshold, we first compute the total of each input's weights. We then abandon all the input states with their sum of the weights is smaller than a carefully selected threshold. By doing so, we are able to reduce the input dimension for about ten times as shown in table 3.1.

	Training Loss	Input Size	Weight Number
Before Pruning	0.5	1368	$>$ 1M
After Pruning	0.8	131+4	5k

Table 3.1: Network Pruning

Figure 3.3 shows how good is our regressor, the KPI we would like to optimize has a mean of 6.49 and a variance of 5.38, the regressor can achieve a mean absolute error of 0.55 on training set and 0.65 on validation set, which we think is good enough as a regressor system.

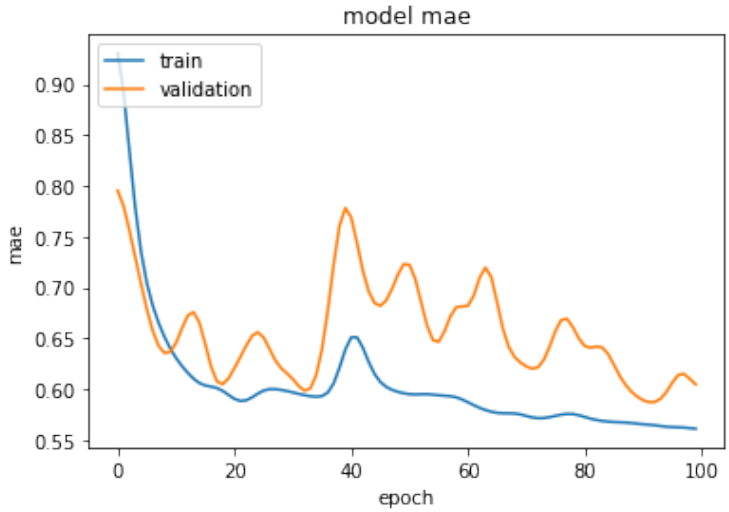


Figure 3.3: KPI regressor loss

3.2.2 State Transfer

Based on the what we have stated in section 2.3, we divide the state into four different parts in 2.3. The $S_{stationary}$ part will not change over time, and A_t is the action from previous time slot, and the state transfer at time t will not change these two parts. As for S_{random}, we will randomly change the state over time with a random noise, so $S_{t+1,random} = S_{t,random} + N_t$ where $N_t \sim \mathcal{N}(0, \alpha * S_{t,random})$, here α is the factor determine how large the noise would be. For the last $S_{predictable}$ part, we assume that all predictable variables in the state follows a Pearson correlation coefficients with the KPI. This coefficient record the relationship between the state variables and the KPI itself, i.e., how likely these states will move in the same direction and for the same degree as the KPI moves. thus the states transfers with $S_{t+1,predictable} = S_{t,predictable} + \alpha * \rho_{K,S_{predictable}} \cdot S_{t,predictable}$ here ρ is the correlation coefficient computed from the whole data set as equation 3.1.

$$\rho_{K,S_{predictable}} = \frac{cov(K, S_{predictable})}{\delta_K \delta_{S_{predictable}}} \tag{3.1}$$

The entire state transfer procedure is presented as equation 3.2

$$S_{t+1} = (S_{stationary}, S_{t,random} + N_t, S_{t,predictable} + \alpha * \rho_{K,S_{predictable}} \circ S_{t,predictable}, A_t) \tag{3.2}$$

Here $\circ$ means pairwise product between the predictable states and the correlation coefficients.

3.3 DDPG

DDPG is a widely used Actor-Critic (AC) framework in dealing with continuous[18] or large[19] state/action spaces. The main components of this system are action NN and value NN.

3.3.1 Action and Value NN

In traditional Actor-Critic frames, as a part of the policy gradient method[1], the output of the actor is a probability distribution. This kind of setting will soon face the curse of dimensionality, so DDPG chooses to be deterministic and avoid potential problems. The actor-network setting thus is changed to

$$\hat{A}_t = \pi(S_t|\theta_\pi) \tag{3.3}$$

The critic networks in DDPG share the same point of the value networks in Sutton's book[1]; they are still an estimate of the Q-function,

$$Q_t = Q(S_t, A_t|\theta_Q) \tag{3.4}$$

Another critical point for DDPG is that except for the normal actor and critic networks, there are also target actor and critic networks, which slowly tracks the actor and critic networks, as shown in figure 3.4. These shadow networks lower the risk of the system diverging or stuck at a semi-optimal soon, for they move much slower than the actor and critic networks, and the update is based on these target networks.

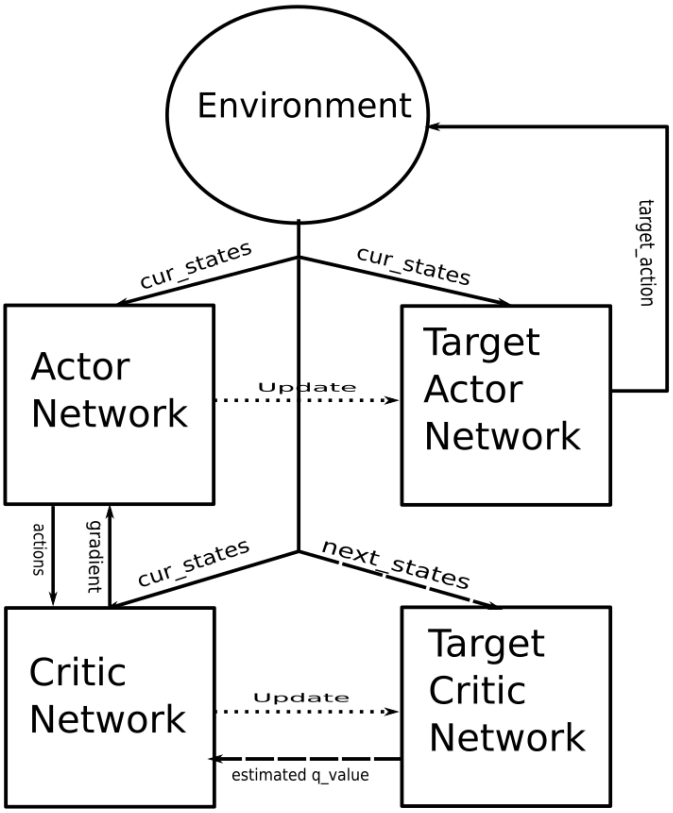


Figure 3.4: A typical DDPG system

3.3.2 Training for the DDPG

The DDPG updates as figure 3.4 shows. First we collect current state S_t from the environment, then pass it to the target actor network to get action A_t. Then feed the action A_t to the environment to get S_{t+1}. Then we are able to update the whole system base on temporal-difference (TD) error $\sigma = r + \gamma * Q(A_{t+1}, S_{t+1}) - Q(A_t, S_t)$, the critic will punish or reward the actor using TD error.

The update of the target network based on the following equation:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\pi'} &\leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi'} \\ where \quad \tau &<< 1 \end{aligned} \tag{3.5}$$

Here τ is the update factor determines how fast the target networks will follow the original networks. This factor τ is usually much smaller than 1.

4. EXPERIMENT RESULTS AND CONCLUSIONS

Training the RL agent efficiently is critical for our system but also a little bit tricky. In this chapter, we will discuss the methods we used during the training, including the exploration method, prioritized experience replay and the use of imitation learning.

4.1 Implementation Details

In this section, we will compare all the method based on two different metrics, **average reward over episodes** and **improving rate**. The improving rate means the ratio of the base stations that can get improved over 5% among all the base stations.

4.1.1 Exploration Strategy

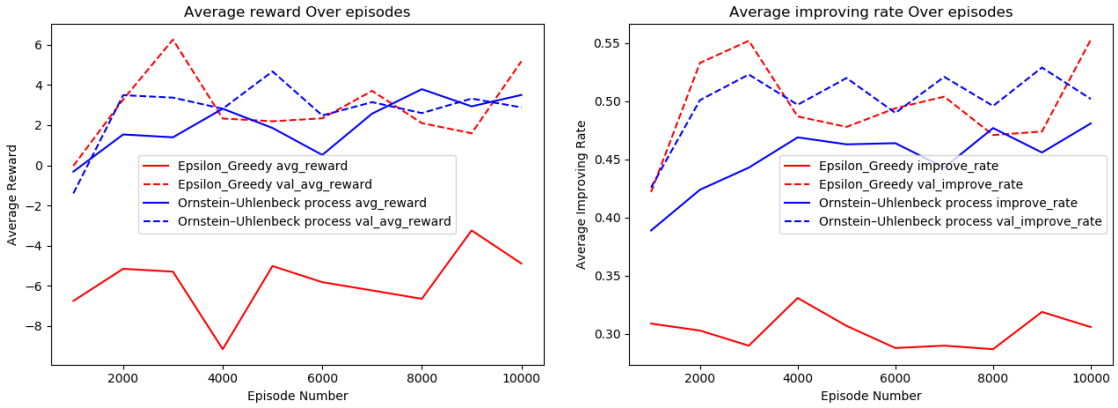
We have two different potential exploration strategies for the agent to explore the action space. One is ϵ-greedy, which is widely used as the standard method in all kinds of RL problems. The other one is the Ornstein-Uhlenbeck (OU) process, which is a random process related to the original data point introduced by the original DDPG paper[18].

We tested our system under both strategies, and the result is shown below,

It seems that the ϵ-greedy suffers from the large action space because it turns to sample from the action space randomly, but our action space has the size of 10^4 choices even with discretization. The OU-process, on the other hand, is a random process center at the current point, which contains randomness, but the randomness has certain restrictions. Thus OU process outperforms the ϵ-greedy method and is the better option for this specific problem.

4.1.2 Prioritized Experience Replay

To learn the strategy more effectively from the replay buffer, the approach to sample from the replay buffer needs to be carefully designed, and a widely used strategy is called prioritized experience replay[20].



(a) Average reward with different exploration strategies

(b) Improvement rate with different exploration strategies

Figure 4.1: The OU process vs ϵ-greedy, it seems the OU process is much better than pure random explore because of the huge action space.

The original paper defines the probability of sampling the point i is,

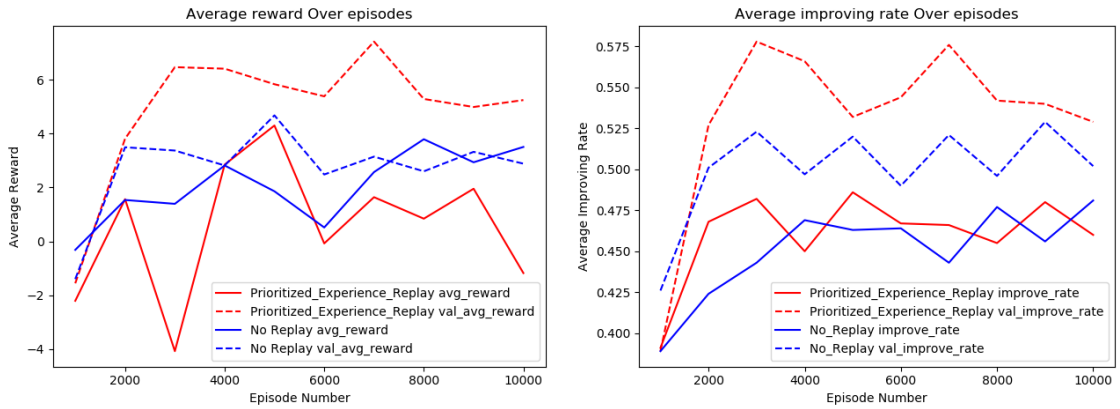
$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{4.1}$$

where α determines how much prioritization is used and p_i is determined by the rank of the ith element which is based on TD-error. However, we would like to acquire the largest accumulated reward, so in our problem, we used the reward R_t to determine the rank.

The result using prioritized replay is shown in 4.2, in which we can see a significant boost in the system's exploring efficiency, the strategy saves more than 1000 episodes on the training set before getting to the best performance, and it gives better performance on the validation set.

4.1.3 Imitation Learning

Imitation learning is a strategy training for the system to imitate the existing strategy or human experts. Theoretically, the imitated agent can get similar performance as the human experts. The imitation learning agent is widely used as the starting point of RL systems and help the systems which lack computational sources to compute more effectively. Our imitation learning agent is also a deep NN that directly maps action from the state.



(a) Average reward with Prioritized experience replay

(b) Improvement rate with prioritized experience replay

Figure 4.2: Prioritized experience replay doesn't really helps the exploration,

$$\hat{A}_t = I(S_t|\theta_I) \tag{4.2}$$

The definition of the strategy looks like actor-network. Instead of guided by the Q-value from the critic network, the imitation learning system treats the action chosen by the experts as the supervised training label. The detailed network structure is in appendix A.2.

In figure 4.3, we can see that with the help of imitation learning, the system is able to converge to the local optimal quicker than the one without imitation learning. That is solid proof that imitation learning can be a good starting point in RL systems.

However, in comparison to Alpha Go and AlphaZero's result[21] where it is evident that the data from the human experts limit the performance instead of helping it in their system. The good thing is that we take advantage of imitation learning without suffering the drawbacks of this method. But in our future work, we need to make this potential problem into consideration.

4.1.4 Algorithm

Summarizing the previous parts, we give our algorithm as below

In this algorithm, all variables with j, d_t means they are sample points below to d_t, and we are taking values from the time instant j when each data point contains data from both j and j-1 time

Algorithm 1: DDPG for BS parameter tuning

Input: minibatch k; stepsize η_a, η_c; exploration factor ϵ; replay period N; budget T; simulator Sim; imitation learning agent I

Output: Current actor and critic parameters θ_π, θ_Q

Initialize the replay buffer H = $\emptyset$, θ_π, θ_Q

Observe the initial state S_0, take action $A_0 = \pi(S_0|\theta_\pi)$

for $t \leftarrow 1$ **to** T **do**

  decay ϵ, η_a, η_c;

  Step Sim, observe $S_t, R_t, done_t$;

  Store $(S_{t-1}, A_{t-1}, R_t, done_t, S_t)$ into H with prioritized experience replay factor;

  if *random* < ϵ **then**

    $A_t = I(S_t|\theta_I)$

  else

    $A_t = \pi(S_t|\theta_\pi) + \epsilon * OU(\pi(S_t|\theta_\pi))$

  end

  Sample from H, $d_t \leftarrow sample(H(\min(k, N)))$

  Compute target-Q value for d_t, $Q_{j,d_t} = Q_{target}(S_{j,d_t}, A_{j,d_t}|\theta_{Q_{target}})$;

  Compute action value for d_t, $A_{j,d_t} = \pi(S_{j,d_t}, A_{j,d_t}|\theta_\pi)$

  Compute TD-error $\sigma = r + \gamma * Q_{j,d_t} - Q(A_{j-1,d_t}, S_{j-1,d_t})$;

  Train critic network based on TD-error $\Delta_{c,d_t} = \sigma \cdot \nabla_{\theta_Q} Q(A_{j,d_t}, S_{j,d_t})$, $\theta_Q \leftarrow \theta_Q + \eta_c \cdot \Delta_{c,d_t}$;

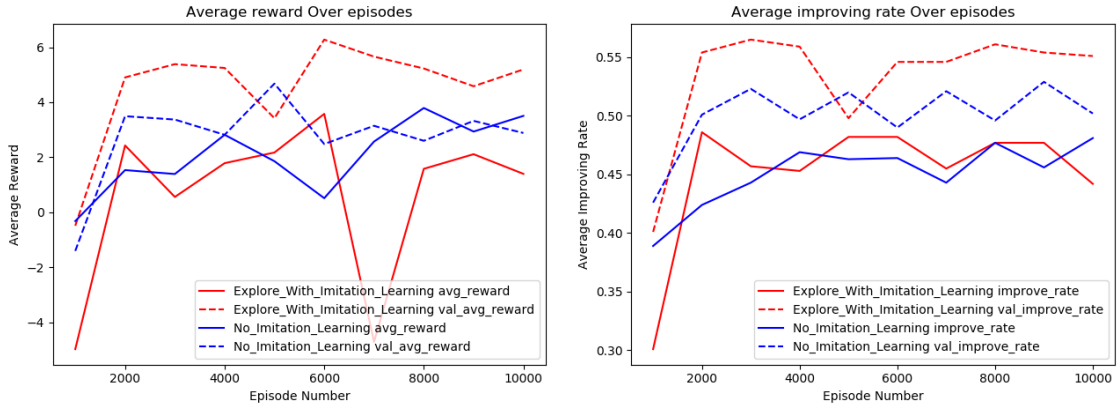
  Compute gradient for actor network $\Delta_{a,d_t} = \nabla_{\theta_\pi} \pi A_{j,d_t}, S_{j,d_t}$;

  Train actor network based on gradient $\theta_\pi \leftarrow \theta_\pi + \eta_a \cdot \Delta_{a,d_t}$;

  Update target networks $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$

end

Save parameters θ_π, θ_Q



(a) Average reward with imitation learning (b) Improvement rate with imitation learning

Figure 4.3: The reward and improve rate with imitation learning, the system converges quicker with imitation learning but and reaches a performance similar to the one without imitation learning

instant.

4.2 Validation Result and Conclusion

Using algorithm 1, we get our final agent for this problem, and we compare the result from the RL agent with the result tuned by the human expert.

The way we validate with the human expert is to roll over the entire dataset, get the real states, and the corresponding tuned data from the human experts. We then feed the states we acquired from the raw dataset into our RL agent and compare their results through the whole dataset.

Method(type)	Improving Rate (Training)	Improving Rate (Validation)	Average Improvement(Training)	Average Improvement (Validation)
Human Expert	50.89%	51.15%	40.67%	39.02%
Imitation Learning	49.78%	50.76%	39.53 %	39.06 %
Imitation+ RL	52.95%	54.16%	40.96%	40.87%
Pure RL	52.95 %	54.16%	40.95%	40.86%

Table 4.1: Result compared to Human Experts

As shown in table 4.1, our method can train agents with performance slightly better than the human experts. Thus, we claim that the Deep RL is capable of optimizing a multi-action continuous

BS parameter tuning system on a single KPI.

Besides the improving rate itself, we spotted other interesting points in this result. We found that for each BS if we are capable of optimizing its performance, we can raise it a lot (over 40 % as shown in the table). However, for nearly half of the BSs, we can not improve them at all. So a potential thought is that can we trade the improvement for improving rate? We can also dig deeper into those base stations who are not able to be optimized, do they follow specific characteristics?

Another interesting result is the difference strategies between the RL agent and human experts. As shown in figure 4.4,

From the figure, we can see that our RL agent tends to choose extreme strategies while the experts will always select more mild actions. We think this difference comes from that the human experts are doing a multi-KPI optimization task, unlike our system.

4.3 Summary and Future Work

Although we achieved human expert level performance in this paper, it is only a starting point of a huge system. As we discussed in the previous section, there remains some mysterious in our project that waits to be unveiled.

One first possible direction is to introduce more KPIs to optimize, which is straight forward. The other possible paths include building a more accurate simulator with a well-estimated state transfer probability or with the help of deep NNs. Or a system that can figure out what is the trade-off between the improvement and the improving rate. We can even find the special characteristics for the bad cells in our system.

We are glad if our work can be an inspiration of any kind for anyone else.

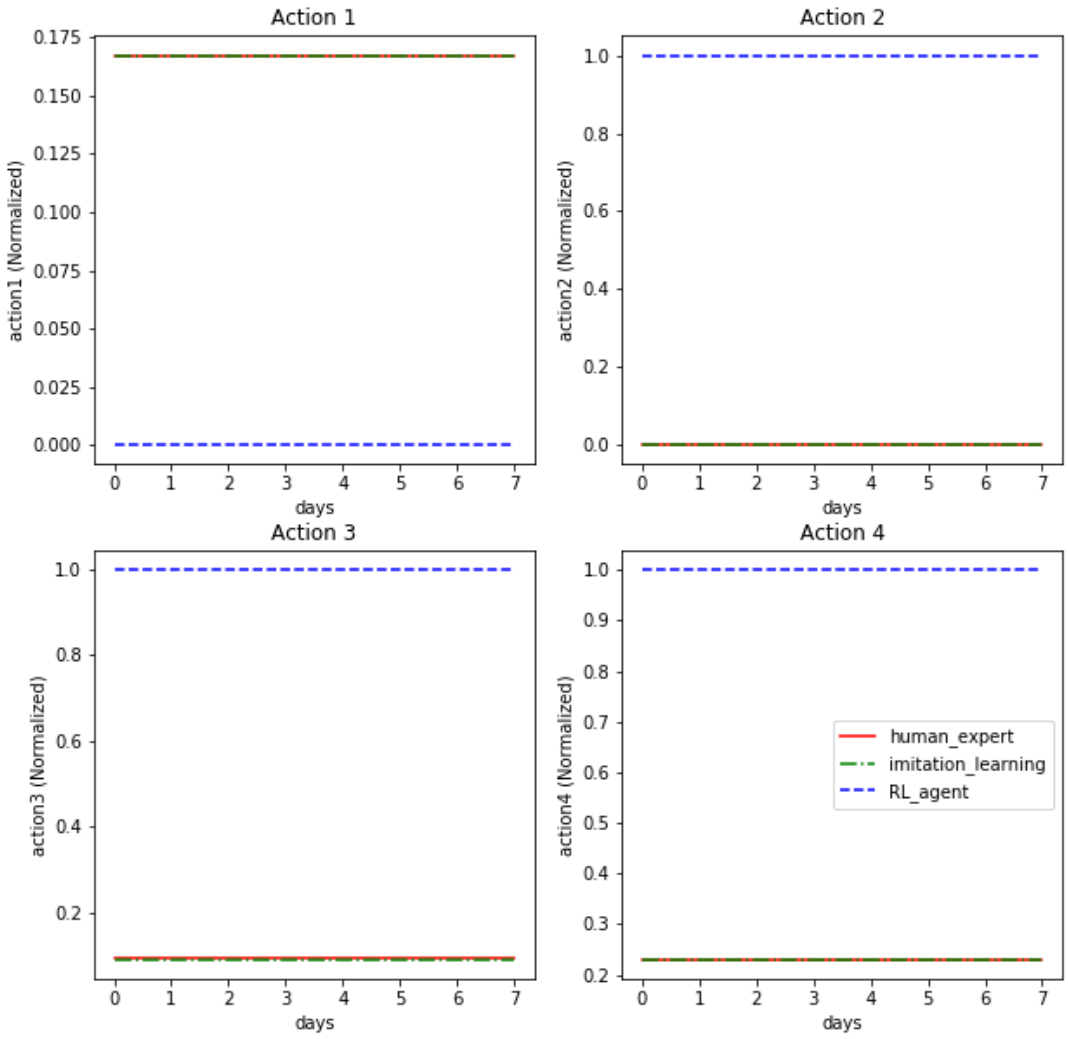


Figure 4.4: Different Strategies Between RL agent and Human Expert

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] L. Venturino, N. Prasad, and X. Wang, "Coordinated scheduling and power allocation in downlink multicell ofdma networks," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 6, pp. 2835–2848, 2009.

[3] F. Rashid-Farrokhi, K. R. Liu, and L. Tassiulas, "Transmit beamforming and power control for cellular wireless systems," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 8, pp. 1437–1450, 1998.

[4] F. Rashid-Farrokhi, L. Tassiulas, and K. R. Liu, "Joint optimal power control and beamforming in wireless networks using antenna arrays," *IEEE transactions on communications*, vol. 46, no. 10, pp. 1313–1324, 1998.

[5] P. Gonzalez-Brevis, J. Gondzio, Y. Fan, H. V. Poor, J. Thompson, I. Krikidis, and P.-J. Chung, "Base station location optimization for minimal energy consumption in wireless networks," in *2011 IEEE 73rd vehicular technology conference (VTC Spring)*, pp. 1–5, IEEE, 2011.

[6] J.-W. Lee, R. R. Mazumdar, and N. B. Shroff, "Downlink power allocation for multi-class wireless systems," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 4, pp. 854–867, 2005.

[7] K.-D. Lee and S. Kim, "Optimization for adaptive bandwidth reservation in wireless multimedia networks," *Computer networks*, vol. 38, no. 5, pp. 631–643, 2002.

[8] M. Andrews, S. C. Borst, F. Dominique, P. R. Jelenkovic, K. Kumaran, K. Ramakrishnan, and P. A. Whiting, "Dynamic bandwidth allocation algorithms for high-speed data wireless networks," *Bell Labs Technical Journal*, vol. 3, no. 3, pp. 30–49, 1998.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep

reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[11] Y. Junhong and Y. J. Zhang, "Drag: Deep reinforcement learning based base station activation in heterogeneous networks," *IEEE Transactions on Mobile Computing*, 2019.

[12] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2017.

[13] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, IEEE, 2018.

[14] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[15] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient model learning methods for actor–critic control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 3, pp. 591–602, 2011.

[16] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[17] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[21] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

APPENDIX A

NETWORK STRUCTURES

A.1 KPI network

The first model is the NN we used to predict the KPI, and we perform their structures in the following tables A.1,A.2[1]:

Layer(name)	Output Shape	Connected to
InputLayer(original)	1348	
Dense(dense1)	128	original
Dense(dense3)	256	dense1
Dense(output)	1	dense3

Table A.1: Regressor with Original State

Layer(name)	Output Shape	Connected to
InputLayer(state)	131	
Dense(dense1)	128	state
InputLayer(action)	4	
Dense(dense2)	128	action
Add(add)	128	dense1, dense2
Dense(dense3)	256	add
Dense(output)	1	dense3

Table A.2: Regressor with States and Action Pruned

[1] In all the structure tables, Dense means a Fully-Connected layer; Connected to means from which layer the current layer get their input. Input layers get data from the outside inputs.

A.2 Imitation Learning

This part shows the structure of the NN that was built to imitate human experts.

Layer(name)	Output Shape	Connected to
InputLayer(state)	131	
Dense(dense1)	300	state
Dense(dense2)	400	dense1
Dense(output)	4	dense2

Table A.3: Imitation Learning Network

A.3 DDPG

This section shows the structure of the DDPG agent, including the actor NN and the critic NN (The target NNs share the same structure as the original ones.)

Layer(name)	Output Shape	Connected to
InputLayer(state)	131	
Dense(dense1)	128	state
Dense(dense2)	64	dense1
Dense(action1)	1	dense2
Dense(action2)	1	dense2
Dense(action3)	1	dense2
Dense(action4)	1	dense2

Table A.4: Action Network[2]

[2]The output dense layers are different because they may have different loss function and activation functions.

Layer(type)	Output Shape	Connected to
InputLayer state	131	
Dense dense1	400	state
InputLayer action	4	
Dense dense2	400	action
Add add	400	dense1, dense2
Dense dense3	300	add
Dense output	4	dense3

Table A.5: Critic Network