An Actor-Critic Algorithm for Modulation and Coding Scheme Selection

Yu Hao, 1831607

June 18, 2019

1 Introduction

In this final project of *Machine Learning* course, I propose a deep reinforcement learning algorithm to make decisions about the selection of modulation and coding scheme (MCS) to improve the throughput of the base station. Numerical experiments indicate the significant performance improvements of the proposed method comparing to traditional statistical methods. In the following parts, I will formulate the problem, introduce two deep reinforcement learning algorithms, and then give some experiments and discussion.

2 Problem Formulation

2.1 System Model

Considering a base station with a single cell serves multi-users for downlink transmission, users may have burst data request. The transmission model is illustrated in Fig. 1 and traffic arrival and departure is shown in Fig. 2. We define the throughput per request per user is

$$S = \frac{M}{t_{fin} - t_{req}},$$

where the M is the packet size of the user want to transmit, the t_{fin} and t_{req} denote the finish time and the request time of the transmission respectively. For example, $UE1$ in Fig. 2 requested a M sized packet at time t_1 and finished at t_2, then its throughput is $\frac{M}{t_2 - t_1}$.

In this system, we are expected to execute Transmission Time Interval (TTI) level scheduling and maximize the objective: the long-term average throughput of the system.

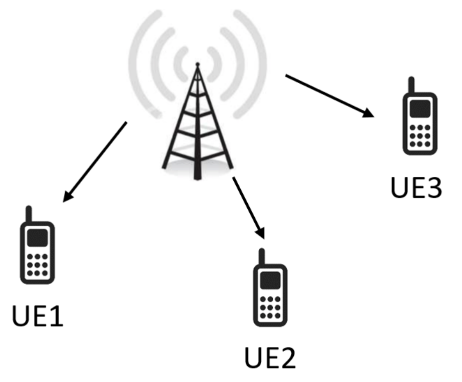

Figure 1: Single cell serves multi-users for downlink transmission

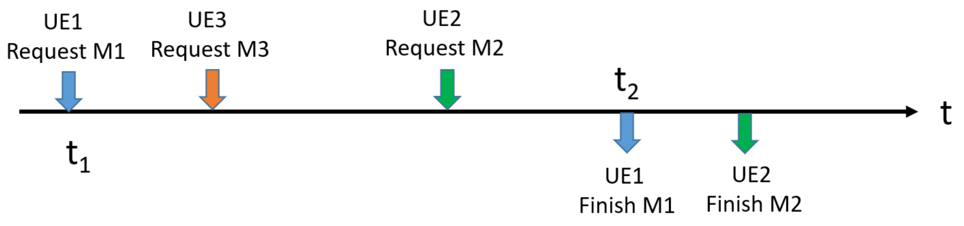

Figure 2: Traffic arrival and departure

2.2 System Simulator

A true base station with multi-users is very complicated with strong randomicity and uncertainty, we cannot train our scheduling policy on an online system. Hence, a system simulator is necessary for model training and testing. The simulator consists of Users Simulator and Base Station Simulator. The Users Simulator is responsible for simulating the addition, deletion, and status updates of users while Base Station Simulator is responsible for simulating the TTI, bandwidth, RBGs and some other parameters of the system.

The detailed description of two simulators are listed in Table 1 and 2.

Table 1: Description and range of parameters for simulating user

Item	Description	Range
ID	The ID of the user in system, starts from 1 and increases consecutively	Positive integer
arr_time	The arrival time of the user, starts from 0	Integer multiples of TTI
buffer	The packet size of user want to transmit	$[10^3, 10^6]$
RSRP	Reference Signal Received Power of the user	$[-120, -90]$
avg_snr	Average Signal-to-noise-ratio of the user	$[1, 31]$
avg_thp	Average throughput of the user	0 to upper bound
CQI	Channel Quality Indication	$[1, 29]$
MCS	Modulation and Coding Scheme	$[1, 29]$
SE	Speed Efficiency	$[1, 6.736]$
prior	Priority of the user in active user list of system	$[0, 6.736]$
sched_rbg	Clarify the RBG is scheduled for the user or not	1 is scheduled and 0 is not

Started from resetting Simulator, the simulator executes the following operations at each TTI.

Table 2: Description and value of parameters for simulating base station

Item	Description	Value
BANDWIDTH	Const, bandwidth of the base station	10^7
RBG_NUM	Const, the number of RBGs	17
TTI	Const, Transmission Time Interval	0.001
UE_ARRIVAL_RATE	The rate of arriving user at a TTI	0.03
CQI_REPORT_INTERVAL	Const, the interval of updating all users' CQI	0.02
PRIOR_THRESHOLD	Const, the throughput threshold of adjusting the priority of the user	10^4
EPISODE_TTI	The simulation time (seconds) of the system for each episode	200

1. Update current time. The sim time will be added by a TTI (0.001 s).
2. Delete users with an empty buffer. Buffer means the left content of user packet.
3. Add new users with the probability of UE_ARRIVAL_RATE. The parameters of a new user follow the equations below.

$$\begin{aligned} ID &:= number\ of\ all\ arrived\ users + 1 \\ arr\_time &:= sim\_time \\ buffer &:= \mathrm{uniform}(10^3, 10^6) \\ RSRP &:= \mathrm{uniform}(-120, -90) \\ avg\_snr &:= RSRP + 121 \\ avg\_thp &:= 0, \end{aligned}$$

   where $\mathrm{uniform(a, b)}$ means the function that take a random number from a to b uniformly.
4. Calculate priority of users. The updating of user parameters is as follows. If the live time of the user is divisible by CQI_REPORT_INTERVAL,

then we need to update its CQI with respect to each RBG. Hence, for the user with each RBG_i, we have

$$\begin{aligned}
CQI_i &:= avg\_snr + \text{randint}(-2, 2) \\
MCS_i &:= CQI_i \\
SE_i &:= \log_2(1 + {MCS_i}^2) \\
prior_i &:= \frac{SE_i}{max(1, \frac{avg\_thp}{PRIOR\_THRESHOLD})}.
\end{aligned}$$

5. Select scheduled users. For each RBG_i, it selects the user with the highest *prior* to be scheduled object. Then, we update the status for selected user.

$$\begin{aligned}
j &:= \arg\max[prior_i] \\
sched\_rbg_{j,i} &:= 1,
\end{aligned}$$

   where j is the user with highest priority for RBG_i, and we set the $sched\_rbg_{j,i}$ to 1 which shows that the $user_j$ is scheduled by RBG_i.

6. Take action. After finishing the user scheduling, the simulator will receive an MCS scheduling action (i.e., the action that we need to offer according to the policy we learned) and then execute the following operations and return the corresponding reward. Suppose that we have M selected users, then, for $user_i$, we have

$$\begin{aligned}
is\_succ_i &:= \begin{cases} 1 & avg\_snr_i + \text{randint}(-2, 2) - MCS_i > 0 \\ 0 & \text{else} \end{cases} \\
rbg\_se_i &:= \log_2(1 + {MCS_i}^2) \\
rbg\_tbs_i &:= 0.9 * BANDWIDTH * \frac{\sum_j sched\_rbg_{j,i}}{RBG\_NUM} * rbg\_se_i * TTI,
\end{aligned}$$

   and the reward of this step is

$$reward := \sum_{i=1}^{M} is\_succ_i * rbg\_tbs_i$$

7. Check if this episode is finished,

$$is\_done := \begin{cases} 1 & sim\_time = EPISODE\_TTI \\ 0 & \text{else} \end{cases}.$$

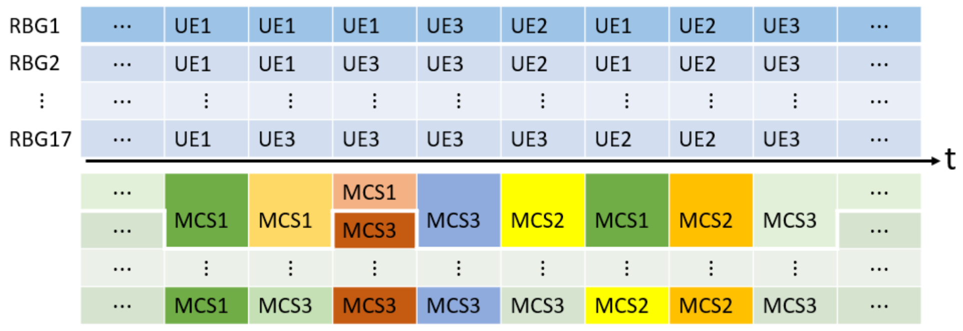

Figure 3: Scheduling: Assign one MCS to the RBGs that are allocated to one UE

2.3 Problem Formulation

What we need to schedule is assign MCS to the Resource Block Groups (RBGs) that are allocated to one user. Take an example, as shown in Fig. 3, the system has 17 individual RBGs with each RBG can only serve one user while one packet of user can be served by several RBGs. It is necessary to mention that the scheduling of RBGs for each user is executed by the system, and we just need to assign a suitable MCS for corresponding RBGs. In addition, the RBGs belonged to one user share the same MCS, hence, we just need to assign MCS for different users.

In our problem, it is necessary to define the state, action, and reward clearly.

In our simulator, the number of active users and scheduled users are dynamic, which is not suitable for building state directly. However, we could create a state from the perspective of RBGs. Whatever the number of scheduled users is, the number of RBGs is constant. Hence, we just need to select the features of scheduled users for each RBG. A scheduled user owns features: buffer, RSRP, avg_snr, avg_thp, [CQI], [SE], [prior], [sched_rbg]. After reducing some duplicate features, we select buffer, RSRP, avg_thp, and CQI as the state of one RBG. Hence, the dimension of state space is RBG_NUM * 4.

However, the action space cannot be selected from the view of RBGs. As mentioned in the simulator, we have one rule: The RBGs belonged to one user share the same MCS. However, we cannot acquire an action result to make sure that RBGs belonged to one user have an equal MCS. Hence, we set the action space from the perspective of scheduled users. The number of scheduled users is upper bounded by the number of RBGs, and each user has 29 MCS levels to schedule. Therefore, the dimension of the action space is RBG_NUM * 29.

As for the reward, the objective is to maximize the expected long-term average reward:

$$\mathbb{E}\left[\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T} r_t\right],$$

where

$$r_t := \sum_{UE_i} is\_succ_i * rbg\_tbs_i.$$

2.4 Baseline

The traditional way to do MCS selection is a statistical method. It assigns the CQIs' arithmetic mean of one user to be the MCS. Assume a user has K RBGs, which means that it has K CQI, and then its MCS can be calculated by

$$MCS := \frac{1}{K}\sum_{k=1}^{K} CQI_k.$$

3 Proposed Algorithms

3.1 General RL Framework

Basically, there are some necessary elements in a general reinforcement learning framework Considering a standard RL setup, which includes an *agent* interacting with an *environment* E in discrete timestamps. At each timestamp t, the *agent* receives an *state* s_t, takes an *action* a_t and receives a scalar *reward* r_t. An agent's behavior is defined by a policy π, which maps states to a probability distribution over the actions $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The environment E may be also stochastic. We model it as a Markov decision process with a state space $\mathcal{S}$, action space $\mathcal{A}$, an initial state distribution $p(s1)$, transition probability $p(s_{t+1}|s_t, a_t)$, and immediate reward function $r_(s_t, a_t)$.

The return from a state is defined as the sum of the discounted future reward:

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$$

with a discounted factor $\gamma \in [0, 1]$. The goal of reinforcement algorithm is to learn a policy which maximizes the expected value from the start state distribution. And we denote the discounted state visitation distribution for a policy π as ρ^π.

How to describe an action's goodness for a specific state? The action-value function (Q value function) is used in many reinforcement learning algorithms. It describes the expected return after taking an action a_t in state s_t, which is formulated as follows.

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i \geq t} \sim E, a_{i > t} \sim \pi}[R_t | s_t, a_t]$$

Typically, we could make use of the recursive relationship known as the Bellman equation:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]].$$

Furthermore, if the policy is deterministic (i.e., one state only maps one action), we can reduce the function to

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))].$$

How to learn the policy from the Q value function? A naive idea is to select the action that could maximize the Q value function at each state. Q-learning [1] proposed the greedy policy by

$$\mu(s) = \arg\max_a Q(s, a).$$

The approximator was not been applied widely due to the non-linear complexity. However, the Q-learning algorithm becomes popular when it used a neural network as function approximators [2, 3].

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[(Q(s_t, a_t | \theta^Q) - y_t)^2\right],$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \theta^Q).$$

The approximator was not been applied widely due to the non-linear complexity. However, the Q-learning algorithm becomes popular when it used neural network as function approximators [2, 3].

3.2 Actor-Critic Method

It's hard to apply the DQN directly to high dimensional or continuous action spaces. It requires optimization of a_t at every timestamp to the find the greedy policy in high dimensional spaces; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Papers [4, 5] proposed an actor-critic approach with deterministic policy gradient (DPG) algorithm. The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. And the critic $Q(s, a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following applying the chain rule to the expected return from the start distribution J with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned}$$

The paper [5] provided a modification to DPG: use neural network function approximators to learn in large state and action spaces, which is called DDPG (Algorithm 1). One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. Obviously, when the samples are generated from exploring sequentially in an environment this assumption no longer holds. Additionally, to make efficient use of hardware optimizations, it is essential to learn in mini-batches, rather than online. The author applied a replay buffer to address the issues. The replay buffer is a finite sized set that stores tuple (s_t, a_t, r_t, s_{t+1}). At each timestamp, the actor and critic network are updated by sampling a mini-batch uniformly from the buffer. Because the DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

The author also uses two tricks to ensure the stability of training: soft target updates and adding noise. Soft target updates constrain the networks to update slowly to avoid divergence. Adding noise helps the actor to explore more actions.

3.3 Soft Actor-Critic

Soft Actor-Critic (SAC) algorithm is proposed by Haarnoja et al. [6]. It is an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework. In this framework, the actor aims to

Algorithm 1: DDPG algorithm

```
1  Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with
     weights $\theta^Q$ and $\theta^\mu$
2  Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q,\ \theta^{\mu'} \leftarrow \theta^\mu$
3  Initialize replay buffer $R$
4  for episode = 1, M do
5      Initialize a random process $\mathcal{N}$ for action exploration
6      Receive initial observation state $s_1$
7      for t=1, T do
8          Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to current policy
             and exploration noise
9          Execute action $a_t$ and observe reward $r_t$ and observe new
             state $s_{t+1}$
10         Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
11         Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$
             from $R$
12         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
13         Update critic by minimizing the loss:
               $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
14         Update the actor policy using the sampled policy gradient:
               $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
15         Update the target networks:
               $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
               $\theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$
16     end
17 end
```

maximize expected reward while also maximizing entropy to succeed at the task while acting as randomly as possible. Experimental results show a better performance of SAC than DDPG.

The pseudocode of SAC is given in Algorithm 2, where ψ is the weights of the state value network, θ is the weights of the Q value network, and the ϕ is the weight of policy network.

Algorithm 2: SAC algorithm

```
1 Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.
2 for each iteration do
3     for each environment step do
4         $a_t \sim \pi_\phi(a_t|s_t)$
          $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
5     end
6     for each gradient step do
7         $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
          $\theta \leftarrow \theta - \lambda_Q \hat{\nabla}_\theta J_Q(\theta)$
          $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
          $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
8     end
9 end
```

4 Experiments

We applied the same neural network: Multilayer Perceptron (MLP) as the actor network and policy network. Each network has two hidden layers with same neurons = 512 (Fig. 4 and Fig. 5 illustrate the actor network and critic network respectively). As for the hyperparameters in RL framework, I choose minibatch size = 512, $\tau = 0.01$, $\gamma = 0.99$. In addition, due to the randomicity of RL training, we execute each algorithm with Monte Carlo simulation for 10 runs to get a more stable result.

Fig. 6 shows the iteration of average reward when training the model.

Both algorithms converge at the end of the episode. However, due to the exploration setting of SAC, it has slower convergence. Fig. 7 presents the testing result of two policies generated by corresponding algorithms with the comparison to baseline. As we can see, both policies perform better than the baseline method.

To acquire a more clear result of the performance, we proposed a relative indicator: *throughput rate*, which is defined as follows.

$$thp\_rate := \frac{\sum_{t=1}^{T} r_t}{\sum_{t=1}^{T} b_t},$$

where b_t indicates the packet size brought from new users at timestamp t. This indicator shows the ratio that the system transmits the buffer successfully. Of course, the theoretical upper bound is 1. A doable policy will increase the *throughput rate* of the system. Fig. 8 shows the experimental result of SAC comparing to baseline. SAC could achieve *throughput rate* above 90% while the baseline could only transmit 50% buffer.

Fig. 9 presents the dynamic changes of active users and scheduled user. The result comes from testing data using the DDPG policy. As we can see, the number of active users changes fast and exists some traffic rush hours. Thanks to the policy we learned, the buffer can be handled timely and no traffic congestion emerged.

5 Conclusion and Discussion

In this project, I applied two off-policed actor-critic based reinforcement learning algorithms to do the selection of MCS. Numerical experiments show the better performance of RL comparing to the traditional method. There are a lot of future works can be studied, I just name a few.

- The scheduling of RBGs. My project just achieves the selection of MCS level, however, the RBG scheduling is important as well.
- The selection of safe policy. A single policy may make an unsafe decision. A novel combination of several policies can reduce the risk.

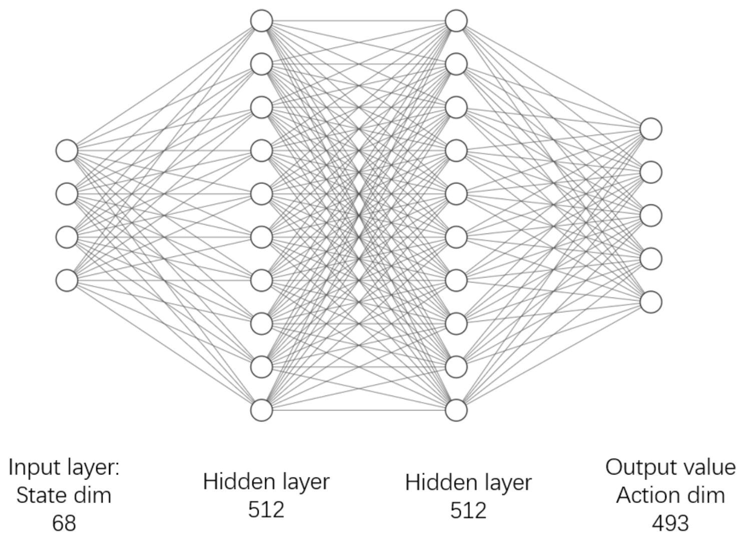

Figure 4: MLP: Actor Network

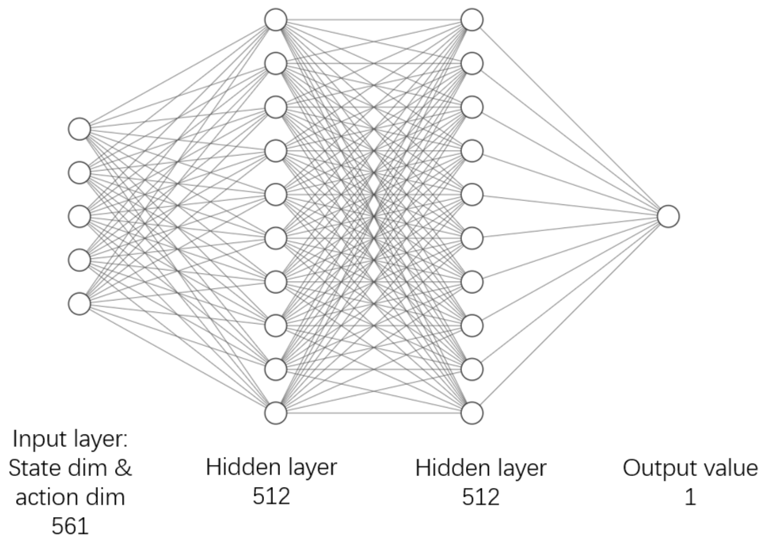

Figure 5: MLP: Critic Network

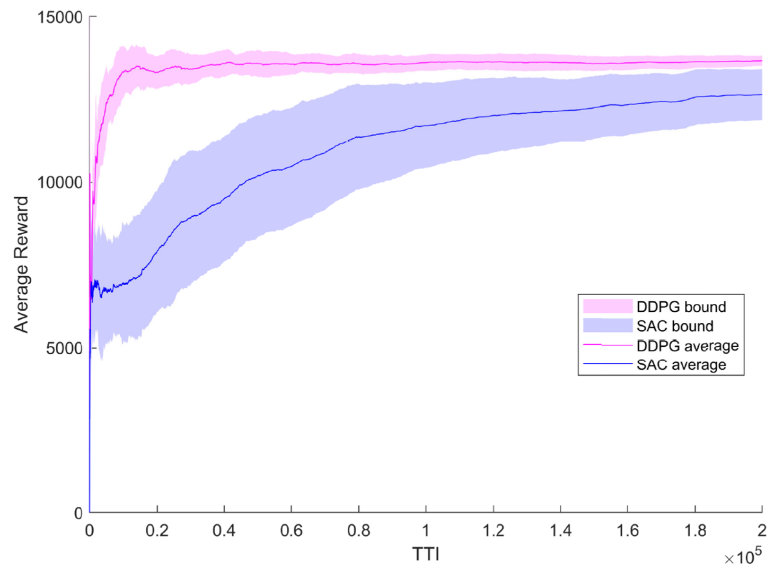

Figure 6: Model training

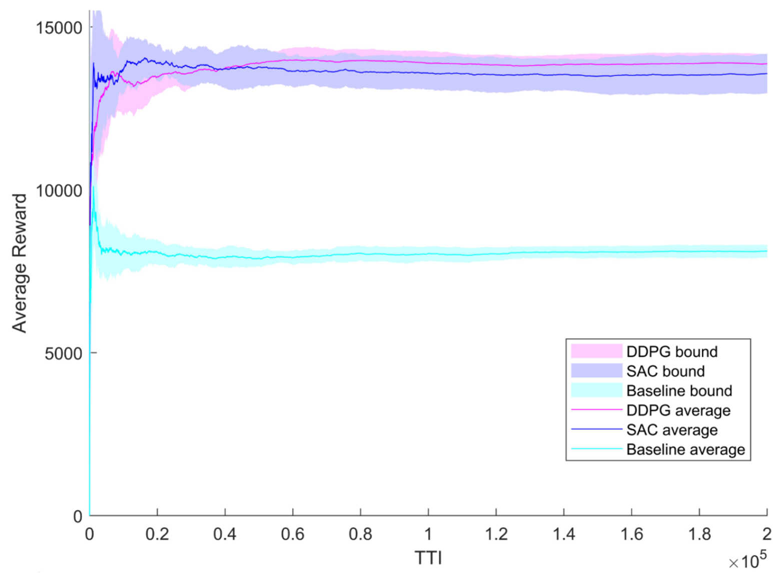

Figure 7: Policy Testing

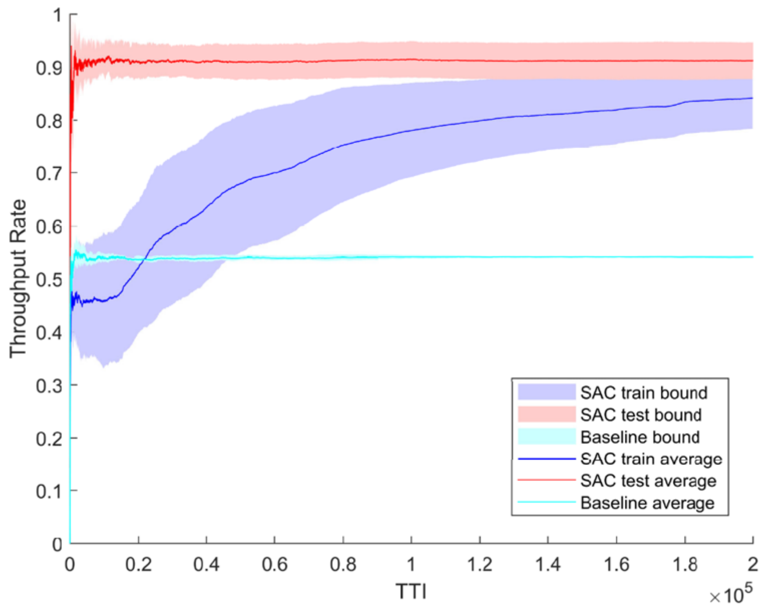

Figure 8: Throughput Rate

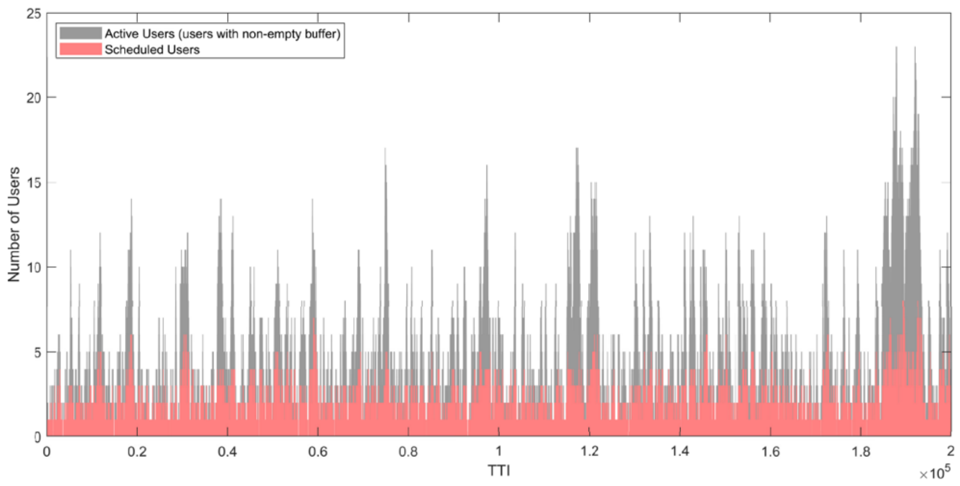

Figure 9: User Scheduling: Active users and scheduled users

References

[1] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[3] 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.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *ICML*, 2014.

[5] 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.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv preprint arXiv:1801.01290*, 2018.