Achieving the Highest Average User Data Rate in Wireless Network using Reinforcement Learning

Qi Cao, Mingqi Yuan, Simon Pun, Yi Chen, Siliang Zeng

April 7, 2020

1 Framework of Reinforcement Learning

Reinforcement learning is learning what to do——how to map situations to actions——so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them [1].Typically, an reinforcement learning system consists of two essential components:interaction environment and agent.With respect to RBG allocation, the scheduling process is discrete owing to its intrinsic motivation. Thus the trajectory of interaction can be formulated as below:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ..., S_{tti}, A_{tti}, R_{tti+1}, ...$$

where $S \in \mathbb{S}$ is the state space, $A \in \mathbb{A}$ is the action space, $R \in \mathbb{R}$ is the reward space. The following section describe the architecture of our reinforcement learning system in detail.

1.1 Interaction environment

The environment gives feedback to the actions made by the agent, including the status of the next state and rewards for the current actions. For such a complex system as communication resource scheduling, the construction of state space is very challenging and valuable.

State In terms of RBG allocation, the state of environment is composed of status of all users in the base station. That is to say, all the attributes are part of this state space, including the known and the unknown. In order to establish an reasonable and quantitative state space, mutiple attributes were carefully seleteced from different layers of communication to characterize the state of interaction environment, and **Table 1** illustrates the details of those attributes.

Attr	Layer	Dim
RSRP	physics	1
Before scheduled buffer		1
History avg rate		1
Olla offset		1
Scheduled times		1
RB CQI	physics	Num of RB
μ(RB CQI)	physics	1
σ(RB CQI)	physics	1

Table 1: Attributes of state

Thus the dimension of state for each user is:

$$dim_{state} = num_{RB} + 7$$

The construction of the subsequent deep neural network is also related to the specific dimension of the state.

Reward With regard to the learning system, the reward signal characterize the optimization target, which motivates the agent to explore any possible solutions. The system aim to maximize the average rate of transmission, to this end, the configuration of reward function must take transmission rate into consideration. The proposed reward function is formulated as below:

$$r_{now} = \frac{r_{before} \cdot N\zeta + s}{N\zeta + \zeta} \quad N = tti_{now} - tti_{before} + 1$$

$$r_{\Delta} = r_{now} - r_{before} = \frac{s}{(N+1) \cdot \zeta}$$

st.

$$reward = \log_{\eta_{avg}}(\sum_{i=0}^{n} r_{\Delta i} + 1)$$

where r is transmission rate, s if effective tb size, ζ is coeffcient. In particular, the log function and η_{avg} were introduced to ensure that the reward function is strictly incremental and locally bounded. A large number of experiments show that the boundedness of reward function will promote the convergence of the model during training.

1.2 Agent

The agent acts on observations of the environment, which is defined in **1.1**. In the following section, the optimization algorithm and network design will be elaborated in detail, respectively.

1.2.1 Deep Deterministic Policy Gradient

The optimization algorithm gives the method of exploration and adjustment, a *Deep Deterministic Policy Gradient* is introduced in this study, which is model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces. [2] The actor network makes actions and the critic network estimates the reward, so let $\mu(s|\theta^{\mu}), s \in \mathcal{S}$ denote the actor and $\mu(s|\theta^{Q}), s \in \mathcal{S}$ as critic, respectively. To summarize, the application of DDPG for RBG allocation is illustrated in the **Algorithm 1**.

Algorithm 1 DDPG for RBG Allocation

Randomly initialize critic network $Q(s, a|\theta^{Q})$ and actor $\mu(s|\theta^{\mu})$ with weights θ^{Q} and θ^{μ}.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replayer buffer B, learning rate τ, Possion distribution $P(X = k) = \frac{\lambda^k}{k!}e^{-k}, k = 0, 1, ...$
for Epoch=1, M **do**
  Randomly initialize the user number K of current epoch
  K users generated from environment
  Receive initial observation state s_1
  for tti=1, T **do**
    Add users to base station according to Poisson distribution
    Execute action $a_{tti} = \beta(s_{tti})$ and observe new state s_{tti+1}
    Receive reward r_{tti} and store the transitions $(s_{tti}, a_{tti}, r_{tti}, s_{tti+1})$ in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from B
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss:$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta_Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^{\mu}}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s_i}$$

    Update the target networks:

$$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q\prime}$$
$$\theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime}$$

This RBG allocation task belongs to discrete action space, however, the DDPG is designed for continuous action space. To make the discrete task become continuous will contibute to the model convergence. In the early stage of proof, algorithms based on stochastic policy, such as Soft-AC [3], had also been tried, but the model often fail to converge.

1.2.2 RBGNet

In the real communication process, the number of users in the base station changes frequently, it means that the state submitted to the agent is not always the same dimension, which leads to great challenges to the network design of the actor and critic. Traditional full connected neural network is widely used in deep Q-learning, but it's limited by its fixed input and output. To address this problem, convolutional neural network is adopted to build the two components, which can accept scalable input when the network is fully convolutional. [4] This network structure designed for RBG Allocation is named **RBGNet**

Backbone **Figure 1** illustrates the architecture of RBGNet. The actor and critic take the same feature extractor to handle the input, which can effectively simplify the design of architecture and adjustment of network parameters. It's worth mentioning that the network is so deep that two residual block is inserted to avoid the gradient disappearing. [5] Also, the very deep network obtains more powerful learning ability to address such a complex problem.

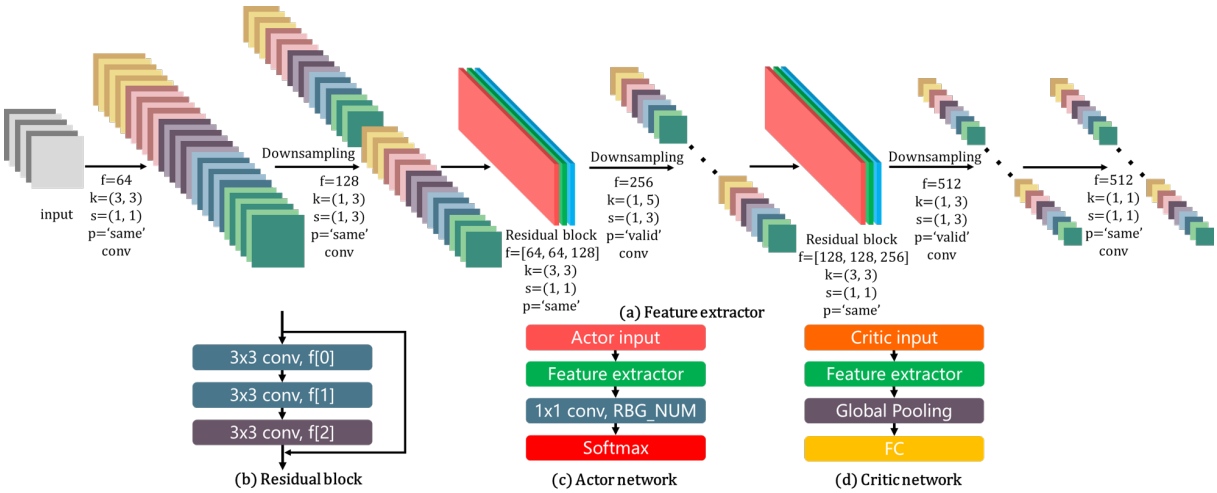

Figure 1: Architecture of RBGNet.The parameters of downsampling layers were configured skillfully based on $dim_{feature}$

.

Remark For the actor network, the input's shape is:

$$dim_{actor\,input} = (batch\,size, dim_{state}, K, 1)$$

where K is the quantity of user, 1 is regarded as channel of image processing. Apply downsampling to the input for three times, the shape of output map is:

$$dim_{output\,map} = (batch\,size, K, 1, F)$$

where F is the quantity of filters in the last convolution layer of feature extractor. As shown above, the third dimension of $dim_{output\,map}$ is 1, which is necessary for the subsequent process, so the parameters of downsampling layer should be configured skillfully based on the dim_{state}.

Then, this map is submitted to the output layer, which is a convolution layer with (1, 1) filters and Softmax activation function, and the Softmax score characterizes the allocation result. Specially, the quantity of filters in the output layer is equal to the quantity of RBG, which aims to make each RGB be allocated by an independent filter. Based on the configuration, each RBG was allocated independently but the prediction is still joint. The final result has nothing to do with the dim_{state} but only related to the quantity of users and RBG.

$$dim_{softmax\,score} = (batch\,size, K, 1, Num_{RBG})$$

For the critic network, the input is the integration of state and action, thus the input's shape is

$$dim_{critic\,input} = (batch\,size, dim_{state} + Num_{RBG}, K, 1)$$

Firstly, the input is submitted to feature extractor to produce intermediate feature. Secondly this map is processed by the Global Pooling layer, which can replace the fully connected layer to accept scalable input. [6] Thus the output is only related to the quantity of filters in the last convolution layer of feature extractor.

$$dim_{output\,map} = (batch\,size, F)$$

Finally, this map is received by a fully connected layer to predict the reward.

1.2.3 Training strategy

The training of reinforcement learning is a difficult and long-time job. In the early stage of the experiment, two models were established to solve the problem of full buffer case and bursty case respectively.

For bursty case, the agent can explore any possible solutions to maximize the average transmission rate since the optimal solutions is uncertain, and the performance of the agent successfully surpasses the *Opportunistic Schedule* after lots of training.

For full buffer case, the agent was hoped to achieve the effect close to that method, but we've found it's difficult to achieve that only by the agent's own exploration. *Opportunistic Schedule* is a fixed method, so it's difficult and impractical for a network with millions of parameters to learn a fixed method by trial-and-error. Howerver, it's clear that the *Opportunistic Schedule* is the optimal solution to maximize average rate, which provides the best experience to guide the agent. Therefore, the **Mixed Experience** was introduced to train the agent, which is composed of the experience both from self-exploration and *Opportunistic Schedule*. With lots of training and *fine-tuning* [7], the agent successfully simulated the *Opportunistic Schedule*.

In order to make the model compatible with both full buffer case and bursty case, a step-by-step training strategy was introduced to address this problem. First of all, the agent is trained with **Mixed Experience**

Algorithm 2 Training strategy

Step 1: full buffer case training
Initialize agent $\mathcal{A}_{RL}$ and $\mathcal{A}_{Oppo}$ respectively.
Initialize $p(\mathcal{A}_{RL}) \leftarrow 0, p(\mathcal{A}_{Oppo}) \leftarrow 0$, where $p()$ is the performance of agent, threshold φ_1.
while $p(\mathcal{A}_{Oppo}) - p(\mathcal{A}_{RL}) > \varphi_1$ **do**
  for Epoch=1, M **do**
    Two agents interact with the environment to generate available experience E^1_{RL} and E^1_{Oppo}.
    Update $\mathcal{A}_{RL}$ with **Mixed Experience** composed of E^1_{RL} and E^1_{Oppo}.
  Update $p(\mathcal{A}'_{RL})$ and $p(\mathcal{A}'_{Oppo})$: $p(\mathcal{A}_{RL}) \leftarrow p(\mathcal{A}'_{RL}), p(\mathcal{A}_{Oppo}) \leftarrow p(\mathcal{A}'_{Oppo})$
  if $p(\mathcal{A}'_{RL}) > p(\mathcal{A}_{RL})$ **then**
    Update $\mathcal{A}_{RL}$: $\mathcal{A}_{RL} \leftarrow \mathcal{A}'_{RL}$
  else
    continue
Save the best agent model $\mathcal{A}^*_{RL}$
Step 2: bursty case training
Initialize agent:$\mathcal{A}_{RL} \leftarrow \mathcal{A}^*_{RL}$
Initialize $p(\mathcal{A}_{RL}) \leftarrow 0, p(\mathcal{A}_{Oppo}) \leftarrow 0$, threshold φ_2
while $p(\mathcal{A}_{RL}) - p(\mathcal{A}_{Oppo}) < \varphi_2$ **do**
  for Epoch=1, M **do**
    Two agents interact with the environment to generate available experience E^2_{RL} and E^2_{Oppo}.
    Update $\mathcal{A}_{RL}$ with E^2_{RL}.
    Update $p(\mathcal{A}'_{RL})$ and $p(\mathcal{A}'_{Oppo})$: $p(\mathcal{A}_{RL}) \leftarrow p(\mathcal{A}'_{RL}), p(\mathcal{A}_{Oppo}) \leftarrow p(\mathcal{A}'_{Oppo})$
  if $p(\mathcal{A}'_{RL}) > p(\mathcal{A}_{RL})$ **then**
    Update $\mathcal{A}_{RL}$: $\mathcal{A}_{RL} \leftarrow \mathcal{A}'_{RL}$
  else
    continue

in full buffer case, which can simulate the behavior of *Opportunistic Schedule*. At this stage, the *fine-tuning* operation should be repeated again and again to get the best performance.

Secondly, using bursty case to train the previous model with low learning rate.Now, the experience is not mixed, which is only from the self-exploration. In a word, the training strategy is full of technique which should be configuration based on the specific situation.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning:An Introduction*. 1998.

[2] 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," *Computer Science*, vol. 8, no. 6, p. A187, 2015.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,"

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[5] K. He, X. Zhang, S. Ren, and S. Jian, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision & Pattern Recognition*, 2016.

[6] M. Lin, Q. Chen, and S. Yan, "Network in network," *Computer Science*, 2013.

[7] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification,"