American Academic Scientific Research Journal for Engineering, Technology, and Sciences

ISSN (Print) 2313-4410, ISSN (Online) 2313-4402

http://asrjetsjournal.org/

A Deep Reinforcement Learning Approach to Queue Management and Revenue Maximization in Multi-Tier 5G Wireless Networks

Elizabeth M. Okumu*

Kabarak University, School of Science Engineering and Technology, P. O. Box Private Bag 20157, Nakuru 20100, Kenya

Email: eokumu@kabarak.ac.ke

Abstract

It is envisioned that 5G systems will increasingly leverage on the network slicing concept to meet the demand of diverse services, each tailored for specific user requirements. In this context, slice admission algorithms that admit slices to the system, that optimize a given objective while ensuring the efficient allocation of resources, are required. Reinforcement learning has been used successfully to implement optimal slice admission policies. But as the 5G wireless network becomes more extensive and intricate, the state and action spaces become large. The efficiency and convergence of reinforcement learning slice admission algorithms is negatively impacted in such a scenario. To improve on this, deep reinforcement learning, a combination of reinforcement learning and deep learning, has been adopted. In this paper, a Deep Q-Learning slice admission algorithm is designed; to this end a utility, was developed. Results show that using the utility as a maximization objective enabled the designed algorithm to (i) optimize the infrastructure provider's revenue while (ii) providing queue management, in terms of queue length and queue delay.

Keywords: Deep reinforcement learning; Reinforcement learning; Network slice; 5G; Slice admission; Resource allocation.

1. Introduction

Fifth Generation (5G) communication networks are expected to be become increasingly heterogeneous by integrating different wireless technologies in a bid to satisfy diverse user demands. In such networks, network entities will be required to make autonomous decisions that optimize resource allocation so as to meet specific user requirements; such as throughput, latency, reliability and efficiency. Machine learning has been successfully used in other fields to solve decision making problems [1]; this success has seen machine learning being applied in the field of wireless networks.

----

* Corresponding author.

In this context, reinforcement learning (RL) has been employed in dynamic decision making in the highly uncertain, time varying and heterogeneous 5G wireless network environment [2]. Reinforcement learning enables a network agent make optimal decisions; this is achieved by the agent interacting with the environment and observing how it responds to its actions. The agent uses states and actions and the response of the environment to its actions to continually modify its actions; this enables the agent independently learn the optimal sequence of actions that optimizes a given objective [1,3]. However, as the complexity and size of the wireless network increases, the state and action spaces become large; this results in the convergence to optimal policy slowing down in reinforcement learning algorithms such as Q-learning algorithms. Additionally, the Q-tables become too large to practically maintain on mobile devices. To overcome these shortcomings, deep reinforcement learning (DRL), a combination of reinforcement learning and deep learning was developed. Applications of DRL in wireless communications include network access and adaptive rate control, proactive caching and data offloading, network security and connectivity preservation, resource sharing and scheduling among others [2]. With respect to resource allocation in 5G systems, the concept of network slicing, which promises improved performance and efficiency, has been widely adopted in literature [4,5]. Accommodating heterogeneous services, using the same infrastructure, on the current monolithic wireless network architecture cannot be accomplished effectively and efficiently. Network slicing overcomes this by dividing the infrastructure into logical networks (slices) each customized to provide the quality of service required for diverse applications. The owners of the physical network resources, known as infrastructure providers, sell the network slices to tenants who then sell their services to end users. Each network slice instance consists of a set of virtual network functions (VNFs) run on the same network infrastructure. VNFs are dynamic, unlike the network infrastructure, which is largely static; this gives VNFs the capability of supporting time-varying application specific service requests. Upon the reception of a slice request, resources are dynamically allocated and associated VNFs launched, i.e. that is orchestrated [6,7]. With network slicing, diverse services with unique QoS requirements can be deployed on the same physical infrastructure in a cost effective, efficient and effective manner. With the advent of network slicing for 5G systems, comes the need for slice admission algorithms. Slice admission should be implemented in manner that efficiently allocates network resources so as to meet predetermined objectives for the infrastructure provider coupled with the specific requirements of the network slice. Machine learning has been applied to slice admission algorithms in literature. In [8,9,10] reinforcement learning based slice admission algorithms that maximize the revenue of the infrastructure provider, are presented. The work in [8] developed a Q-learning algorithm that admits slices, in a multi-tier 5G wireless network, that maximize the revenue while satisfying resource constraints, in terms of the maximum system capacity. The authors in [9] represent the resource constraints in terms of an admissibility region, which an admitted slice must lie within; based on this, a Q-learning slice admission algorithm that optimizes revenue was developed. While in [8,9] the service requirements of the slice requests are in terms of capacity, and the requests are serviced in a first-in-first-out (FIFO) manner, in [10] latency requirements are considered and priority based slice admission is applied. A RL based slice admission policy, that maximizes revenue, by admitting slices depending on their priorities (highest priority first), was developed. Application of DRL slice admission algorithms for revenue maximization is demonstrated in [11], where the work done in [9] is extended by incorporating DRL techniques. The authors in [12] proposed a DRL based slice admission algorithm that maximizes a utility, which they defined as the immediate reward minus the queuing delay cost. In this paper, the work done in [8] is extended by incorporating DRL techniques. A utility, used as an optimization objective for the DRL based slice admission algorithm, is proposed. Optimization of the developed utility resulted in the designed Deep Q-learning slice admission algorithm having the capability of maximizing the revenue, while at the same time managing the queue lengths and queue delays.

2. An Overview of Deep Reinforcement Learning

Machine learning, is a subset of artificial intelligence, which focuses on building algorithms that use training data or past experience to automatically improve their performance at a given task. Machine learning algorithms can be divided into three broad categories; supervised learning, unsupervised learning and reinforcement learning [13]. The aim of supervised learning is to learn the optimal mapping function that maps inputs into outputs; this is accomplished by training the algorithm using labeled input and output data. Examples of supervised learning include classification, regression and neural networks. Unsupervised learning algorithms analyze unlabeled data with the aim of identifying patterns within the data set; clustering is an example of supervised learning. In reinforcement learning, an agent learns by taking actions in an environment, observing the rewards (or penalty) received, and then automatically modifying its strategy to achieve the optimal policy, which is a sequence of reward maximizing actions for every state [14]. Deep reinforcement learning, more specifically deep Q-learning, combines reinforcement learning and deep learning. In this section, the fundamentals of reinforcement learning (RL) and deep reinforcement learning (DRL) are presented.

2.1. Reinforcement Learning

In reinforcement learning, a Markov Decision Process (MDP) [15] is used in problem formulation; in MDP the system is modeled as (i) a finite set of states, $s \in S$, (ii) a finite set of actions, $a \in A$, (iii) probability of transitioning from state s to state s' $(s, s' \in S)$ when action $a \in A$ is executed and (iv) an immediate reward, r, received after action a is performed. The agent exists in an environment defined by the state space S, and in each discrete time step t, the agent performs action a_t while in state s_t it receives a reward r_t which is the immediate value of the state-action transition, as shown in Figure 1(a). The policy $\pi : S \rightarrow A$ is a state to action mapping. In MDP problem formulation, the aim is for the agent to maximize the cumulative reward. To this end, for a finite time horizon and discounted MDP, the value function that is used to measure the worth of a policy is defined as follows

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \forall s \in S$$

$0 \leq \gamma < 1$ is the discount factor. The optimal policy π^*, enables an agent select the optimal action a_t, given a current state s_t ; this can be achieved by the agent learning the optimal value function given by [3]

$$V^*(s) = \max_{a_t}[r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})]$$

2.1.1. Q-Learning

The most popular and effective reinforcement learning technique in literature is Q-learning. Q-learning is model free in that it does not use transition probabilities or the value function $V^*(s)$ defined above. In Q-learning, the optimal policy is realized by determining the optimal values of the Q-function, $Q(s,a)$, defined as

$$Q(s,a) = r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})$$

The Q values are stored in a table, the Q-table, with each state-action pair having a separate entry. The Q-table is updated in an iterative manner as follows. The agent, while in the present state s, executes an action a, receives reward r and moves to the next state s'. The entries in the Q-table are then updated using the following rule, where $Q_n(s,a)$ denotes the learned value after the nth iteration;

$$Q_n(s,a) \leftarrow Q_{n-1}(s,a) + \alpha_{n-1}\left[r_n + \gamma \max_{a'} Q_{n-1}(s',a') - Q_{n-1}(s,a)\right]$$

$\alpha_n \in [0,1]$ is the learning rate which has to satisfy the conditions $\sum_{n=0}^{\infty} \alpha_n = \infty$ and $\sum_{n=0}^{\infty} \alpha^2 < \infty$ [16]. The optimal state-action pairs, i.e. the actions that optimize the Q-value for each state, constitute the optimal policy.

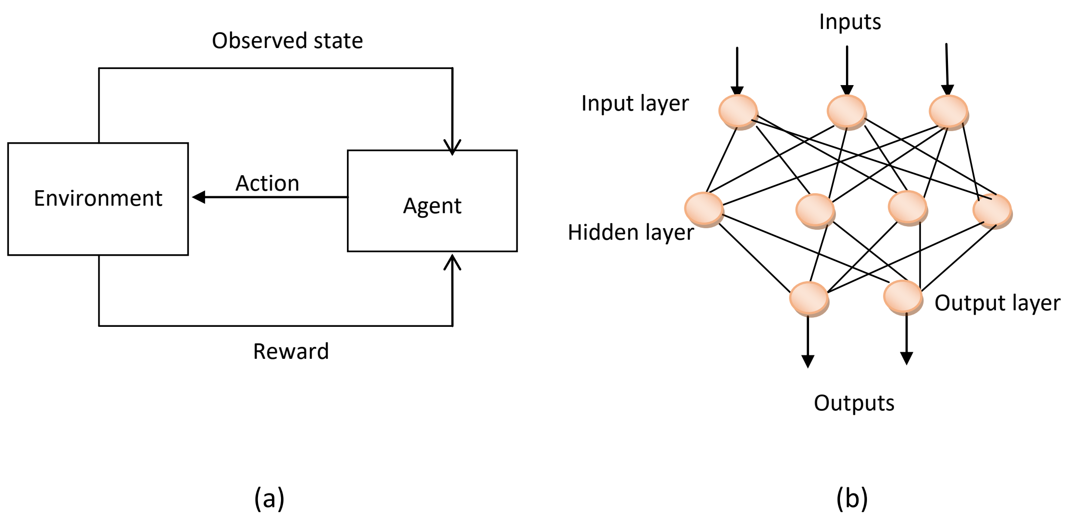


Figure 1: (a) Reinforcement learning (b) Artificial neural network.

2.2. Deep Reinforcement Learning

Deep reinforcement learning combines reinforcement learning and deep learning; it leverages on the capabilities of deep learning to improve the performance of reinforcement learning algorithms.

2.2.1. Deep Learning

Deep learning techniques and algorithms have the ability of learning from large amounts of data; they accomplish this by utilizing a Deep Neural Network (DNN) which gives them the ability to extract high-level abstract features of raw data. Most DNNs are based on Artificial Neural Networks (ANNs), which are inspired by the human brain [17]. An ANN is made up of network units (neurons) in three interconnected layers, the input, hidden and output layers as shown in Figure 1(b). The input layer neurons receive input data, compute it, and forward the result to the hidden layer units. The computed outputs of the hidden units are used as inputs to the output layer neurons. Each unit computes a single output based on a weighted sum of its inputs and an activation function. Some examples of activation functions used in neural networks are the Sigmoid and Relu activation functions [3], shown in Figure 2. Learning in neural networks involves determining the weight values (w_i) that enable the ANN perform the task that it was designed for. The appropriate weight values are obtained by training the ANN using the backpropagation algorithm and a set of training data. The training data set consists of input data and the expected (correct) output, i.e. the target value. Backpropagation employs gradient descent in an effort to minimize the loss function between network outputs, obtained by feeding forward the input data, and the corresponding target values. The gradient of the loss function is calculated, and then propagated back across all neurons and is used to adjust the neuron weights [1] [3]. A DNN is a neural network

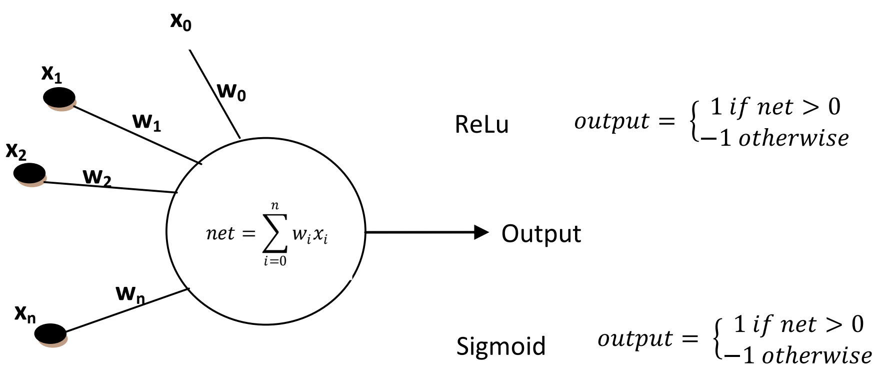

x_0
x_1
x_2
x_n
w_0
w_1
w_2
w_n
$net = \sum_{i=0}^{n} w_i x_i$
Output
ReLu
$output = \begin{cases} 1\, if\ net > 0 \\ -1\, otherwise \end{cases}$
Sigmoid
$output = \begin{cases} 1\, if\ net > 0 \\ -1\, otherwise \end{cases}$

Figure 2: Relu and Sigmoid activation units.

with one or more hidden layers. There are two basic types of DNNs; the Feedforward Neural Network (FNN) and the Recurrent Neural Network (RNN). In FNN information only moves in one direction, forward; from the input units, through the hidden layers units and then the output units. On the other hand, in RNNs, information cycles through loops; current inputs and what was learned from previous inputs is used in decision making. Among the FNNs, the Convolutional Neural Network (CNN) is the most well known model because of its application in image and speech recognition and processing [18].

2.2.2. Deep Q-Learning

Deep Q-Learning is a DRL model that uses a Deep Q-Network (DQN), which is basically a DNN, to approximate the Q-function, in place of the Q-Table, as shown in Figure 3. The function of the DQN is to accept a state as its input and then calculate the Q-values for every action in the action space for that state. The present state, s, is passed through the neural network and then the maximum output Q-value (predicted Q-value) is determined. The action corresponding to the predicted Q-value is executed, and the reward, r, from the environment and the next state s' are observed. The state s' is then passed through the DNN, and the target

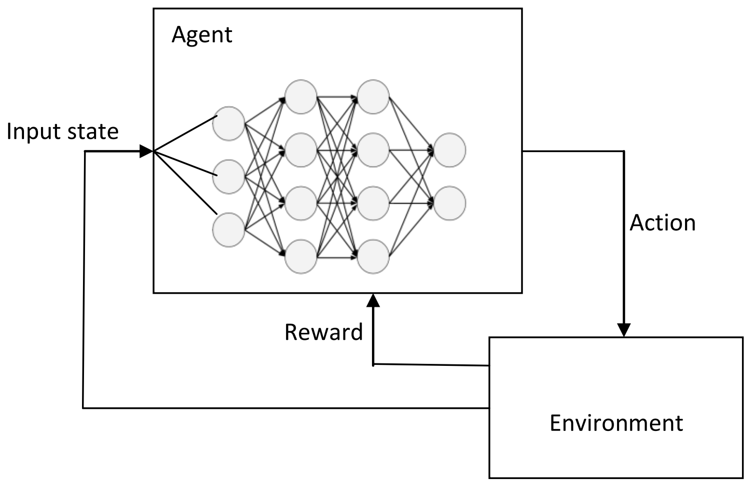


Figure 3: Deep Q-learning.

Q-value required for training the neural network is calculated using $r + \gamma \max Q(s', a')$ where $\max Q(s', a')$ is the maximum Q-value from among the Q-values calculated by the neural network with s' as the input state. The following techniques are used to improve the performance of Deep Q-learning algorithms [19]

- Fixed target Q-Network: During the training process, the weights of the deep Q-network are updated frequently. Using the same network to calculate the target Q-values results in constantly changing values being used to update the Q-network; this introduces instability in the algorithm. This issue can be addressed by using a separate target Q-network; the weights of the target Q-network are updated every so often with the weights of the main (primary or policy) Q-networks.
- Experience Replay: The agent's experiences at each time step are stored in a replay memory. Each experience includes the current state, action taken, reward received and the next state, i.e. (s_t, a_t, r_t, s_{t+1}). The algorithm randomly selects samples from the replay memory to train the neural network. Training the DNN using the predicted Q-values and the calculated target values will result in inefficiencies in the learning process; this is due to the correlation between the Q-values and the target values. Using random samples for the replay memory breaks this correlation.

3. Deep Q-Learning Slice Admission Algorithm

3.1. System Model

The network model used is that of a multi-tiered 5G network as in [8]. A single macro-cell, with a capacity (in bits per second) of C_{Mac}, is overlaid with B non overlapping small cells; the small cells have identical coverage area and capacity, C_{Sc}. It is assumed that a user, at any given time, is associated with a single base station. The users are also assumed to be uniformly distributed in the macro-cell. Three service types will be considered; ultra-reliable low latency communication (uRLLC), enhanced mobile broadband (eMBB) and internet of things

(IoT) services as in [8]. The capacity requirement for the uRRLC and eMMB users is C_u and C_e respectively; an IoT user, defined as a group of d_i IoT devices, requires a capacity of C_i. IoT and eMMB users only receive network service via the macro-cell base while uRRLC users, are serviced by the small cell that they are located in. The network is assumed to be owned and operated by an infrastructure provider who provides network slices to tenants; the tenants in turn provide services to the end users. A network slice is characterized by the tuple $[m, n, t, \rho_m]$ where a) $m \in [1, 2, 3]$ are the supported service types with IoT, uRRLC and eMMB represented by $m = 1, 2\ and\ 3$ respectively b) n is the size of the slice, i.e. the number of users the slice should accommodate c) t is the number of time units the slice is requested for and d) $\rho_m$is the price per unit time per user, for service type m, that the tenant has to pay for acquiring the slice from the infrastructure provider.

3.2. Algorithm Description

The system is modeled as a set of distinct states, $s \in S$, where a state is defined as an $3 \times N$ matrix, where N is the maximum size of a network slice. The rows represent the service type, while the columns represent the size of a network slice. Therefore, the element s_{mn} denotes the number of slices of size n and type m that are present in the system. Incoming slice requests, of the form $[m, n, t]$, are placed in a queue, Q_{mn}, depending on their type and size. The Deep Q-learning algorithm retrieves the slice requests from the queues in a FIFO approach, and allocates them appropriate network resources if they are available. Once the slice request is processed, it is removed from the queue. The possible actions, $a \in A$, defined as (a_1, a_2, a_3, R), where a_m is an N-tuple $(a_{m1}, a_{m2}, \ldots, a_{mN})$; the nth element a_{mn} corresponds to the action of accepting a slice of size n of service type m. If appropriate resources for a slice request are not available, then the request will remain in the queue; this is represented by the action R. The Q-learning algorithm is designed to maximize the utility defined as follows;

$$\varrho = 1 + \log_e[nt(\rho_m + \varphi)]$$

with

$$\varphi = \begin{cases} c_m\left(T_{age.m} - \tau_m\right) & T_{age.m} > \tau_m \\ 0 & T_{age.m} \leq \tau_m \end{cases}$$

where (i) $T_{age.m}$ is the queuing delay of a slice request of type m (i.e. the time the slice request has spent in the queue) (ii) τ_m is the service type dependent tolerable delay and (iii) c_m is the cost associated with delaying a slice request in the queue for a period more than the tolerable delay. Each time a slice is admitted into the system, the defined utility is gained by the algorithm; at the same time, the infrastructure provider earns a revenue of $tn\rho_m$. The algorithm maximizes the utility subject to the following capacity constraints

$$C_i \sum_{n=1}^{N_1} ns_{1n} + C_e \sum_{n=1}^{N_3} ns_{3n} \leq C_{Mac} \qquad C_u \sum_{n=1}^{N_2} ns_{2n} \leq BC_{Sc}$$

A DQN is implemented to achieve utility maximization; the DQN accepts the state matrix elements as inputs, therefore the number of inputs to the DQN is mn. The outputs of the DQN, i.e. units in output layer, represent all possible actions for a given state and are $N \times 3 + 1$ in number. The maximum network slice size was set to $N = 2$, making the number of inputs and outputs 6 and 7 respectively. The DQN has two hidden layers; the first and second hidden layers have 15 and 25 neurons respectively. The activation function used in the hidden layers is the Sigmoid function while the ReLu function is used in the output layer. The Deep Q-learning algorithm, with experience replay and target Q-network, is described in Table 1.

Table 1: Deep Q-Learning Algorithm.

```
Initialize replay memory, D, with a capacity of M
Initialize the policy Q-network, $\boldsymbol{Q}$, with random weights $\boldsymbol{W}$
Form the target Q-network, $\widehat{\boldsymbol{Q}}$ by cloning the policy network, i.e. $\boldsymbol{W'} = \boldsymbol{W}$
for episode = 1 to E
        for each time step
                Select an action $a_t$ via exploration or exploitation
        Perform the action $a_t$, observe the immediate reward $r_t$ and the next state $s_{t+1}$
        Store the experience, $(s_t, a_t, r_t, s_{t+1})$, in the replay memory D
        Select a random batch of J samples from D
        for  j = 1 to J
Pass $s_j$ through the policy Q-network; select from the output Q-values, the Q value that corresponds to the sample action, i.e. the predicted Q-value $\boldsymbol{Q}(s_j, a_j)$
Pass $s_{j+1}$ through the target Q-network to calculate the target Q-value
                $r_t + \gamma \max_{a_{j+1}} \widehat{\boldsymbol{Q}}(s_{j+1}, a_{j+1})$
Calculate the loss between predicted and target Q-values
                $\left[r_t + \gamma \max_{a_{j+1}} \widehat{\boldsymbol{Q}}(s_{j+1}, a_{j+1}) - \boldsymbol{Q}(s_j, a_j)\right]^2$
Back propagate the loss and update the weights $\boldsymbol{W}$
                end for
                After every $x$ time steps set $\boldsymbol{W'} = \boldsymbol{W}$
        end for
end for
```

The epsilon-greedy strategy was used to balance exploration and exploitation. The value of epsilon was initialized to $\boldsymbol{\epsilon = 1}$ and a decay rate of $\boldsymbol{\epsilon_D = 0.001}$ was used. When it is time to select an action, a number between 0 and 1 is randomly selected and compare to $\boldsymbol{\epsilon}$; if it is $< \boldsymbol{\epsilon}$ the random action is selected i.e. exploration. Otherwise, if it is $> \boldsymbol{\epsilon}$ then the action that corresponds to maximum predicted Q-value for that state is selected i.e. exploitation. Epsilon is updated every episode as follows $\boldsymbol{\epsilon = 1 - (E - 1)\epsilon_D}$. The backpropagation algorithm and updating of the policy Q-network weights is implemented as described in [3]. The target Q-network weights are updated after every $\boldsymbol{x = 100}$ time units. The capacity of the replay memory

D was set to $\boldsymbol{M = 10}$, and the random batch that was selected from D for training purposes was $\boldsymbol{J = 5}$. The discount factor was set at $\boldsymbol{\gamma = 0.1}$.

4. Simulation and Results

In this section, the performance of the Deep Q-Learning algorithm is evaluated via simulations. The simulation parameters used are as follows; the number of small cells is set at $B = 3$, and the cell capacities are related as $C_{Mac} = 12C_{Sc}$. The capacity requirements for the network slices are set as $C_u = C_{Sc}/4$, $C_e = 1.5C_u$ and $C_i = C_u/2$. The arrival rates for IoT, eMMB and uRLLC slice requests are 0.2, 0.35 and 0.35 respectively, while the cost per unit time are related as follows, $\rho_2 = 4\rho_1$and $\rho_3 = 3\rho_1$. The tolerable delay and delay cost were set as $\tau_1 = 75$, $\tau_2 = 30$, $\tau_3 = 60$ and $c_1 = 1/32$, $c_2 = 1/20$, $c_3 = 1/40$ respectively. The number of queues is 6, and each queue is stores a maximum of 100 slice requests. The performance of the Deep Q-Learning slice admission algorithm is compared to 1) a Q-learning slice admission algorithm, implemented as in [8], 2) an algorithm that admits slices in a random manner. In Figure 4 a) and b) the average utility per slice request and cumulative revenue per slice request are plotted respectively.

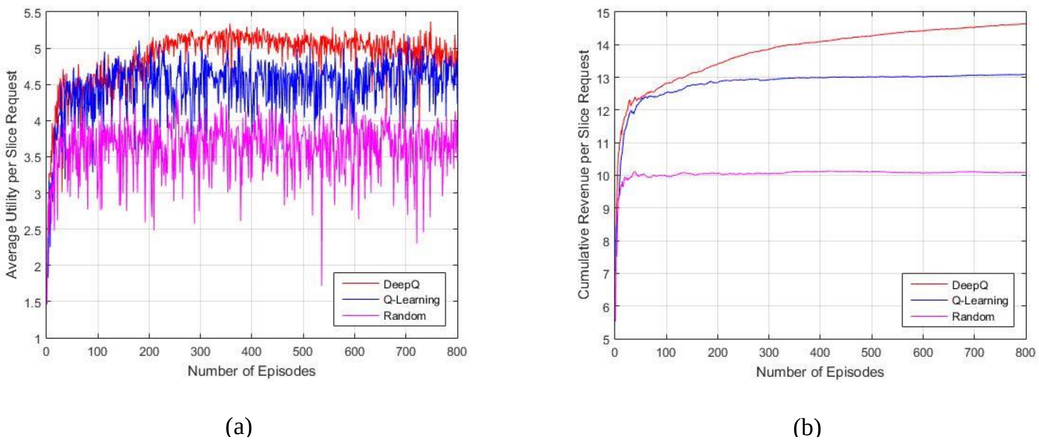
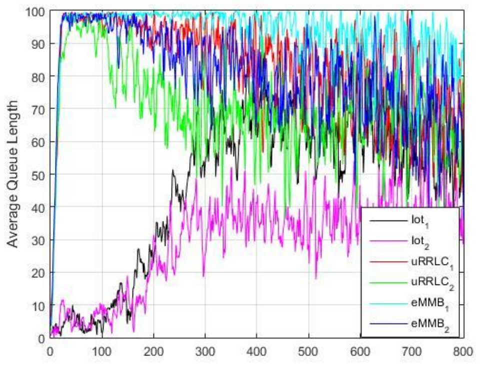
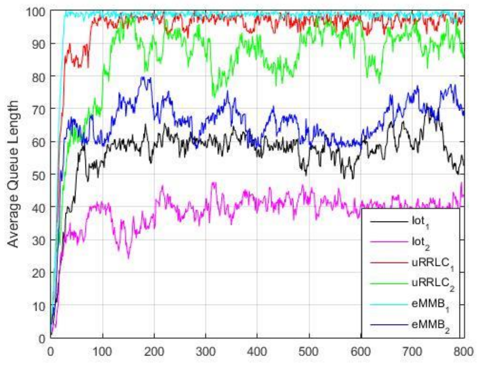


Figure 4: a) Utility and b) Revenue against number of episodes.

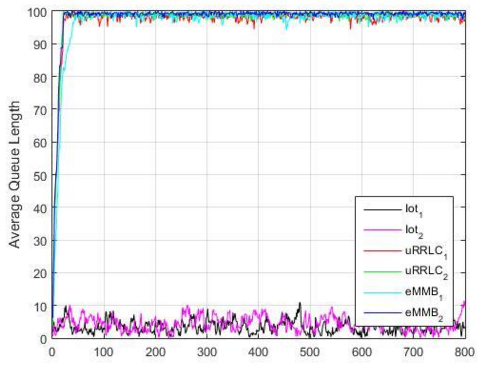
The utility, ϱ, takes into account the revenue to the infrastructure provider, $nt\rho_m$ and the cost of delaying a slice request, i.e. keeping the slice request in the queue over its threshold delay value, φ; therefore, optimizing ϱ results in revenue maximization. From Figure 4, it is observed that the Deep Q-learning algorithm outperforms both the Q-learning and random algorithms. The Q-learning algorithm considers state transitions, which the random algorithm does not, resulting in the Q-learning algorithm having better performance. The Deep Q-learning algorithm additionally uses historical data, stored in the memory replay, to train the network weights; this results in its observed superior performance. In Figure 5 a), b) and c) the average queue lengths for Deep Q-learning, Q-learning and random algorithms respectively are plotted. The queue lengths for the random algorithm are primarily determined by the arrival rate of the slice requests; this is because the slices are admitted in a random manner. Since the arrival rate for the IoT slice requests is less than that for eMMB and uRRLC slice requests, the IoT queue lengths are the shortest, averaging values of below 10, as seen in Figure 5 c), while those for eMMB and uRRLC are consistently equal to the maximum queue length of 100.



(a)



(b)



(c)

Figure 5: Average queue length for a) Deep Q-Learning b) Q-Learning and c) Random Algorithms.

Both the Q-learning and Deep Q-learning algorithms use the utility ϱ as the maximization objective; as observed earlier ϱ is dependent on φ; φ on its part depends on $T_{age.m} - \tau_m$. Therefore, the longer the delay in processing a slice request results in an increase in φ, and a corresponding increase in ϱ; this increases the probability of the slice being admitted. The effects of these are observed in Figure 5 a) and b) where it is noted that the DRL and RL algorithms have queue management capability; in general there does not exist such a large difference in queue lengths, as observed for the random algorithm. From Figure 5 a) it can be seen that the DRL algorithm maintains the queue lengths between 30 and 90, after 300 episodes. On the other hand, the RL algorithm maintains the queue lengths between 40 and 100, after 100 episodes. In Figures 6 (a), (b) and (c) the normalized delays for Deep Q-Learning, Q-learning and random algorithms are plotted respectively. The normalized delay is calculated as follows; $(T_{age.m} - \tau_m)/\tau_m$. For the random algorithm, it is observed in Figure 6 c) that the normalized delays for IoT service requests have the lowest values, of approximately zero, in comparison to that of eMMB and uRLLC slice requests, whose values range from 1 to 25. From Figure 6 a) and b) we observe that the Deep Q-learning algorithm maintains the normalized delays between 2 and 15 after 300 episodes while for the Q-learning algorithm it is between 2 and 20 after 100 episodes. It is observed from Figure 5 a) and 6 a) that after 80 episodes, queue length and queue delay management kicks in for Deep Q-learning slice admission algorithm; from Figure 5 b) and 6 b), this happens after 100 episodes for the RL slice admission algorithm. The Deep Q-learning algorithm outperforms the Q-learning algorithm because the knowledge gained regarding the system and the environment is used in queue length and queue delay management; after 80 episodes, the queue lengths and normalized delay values continue reducing. This is not the case for the Q-learning algorithm, since after 100 episodes, the queue lengths and normalized delays remain approximately constant.

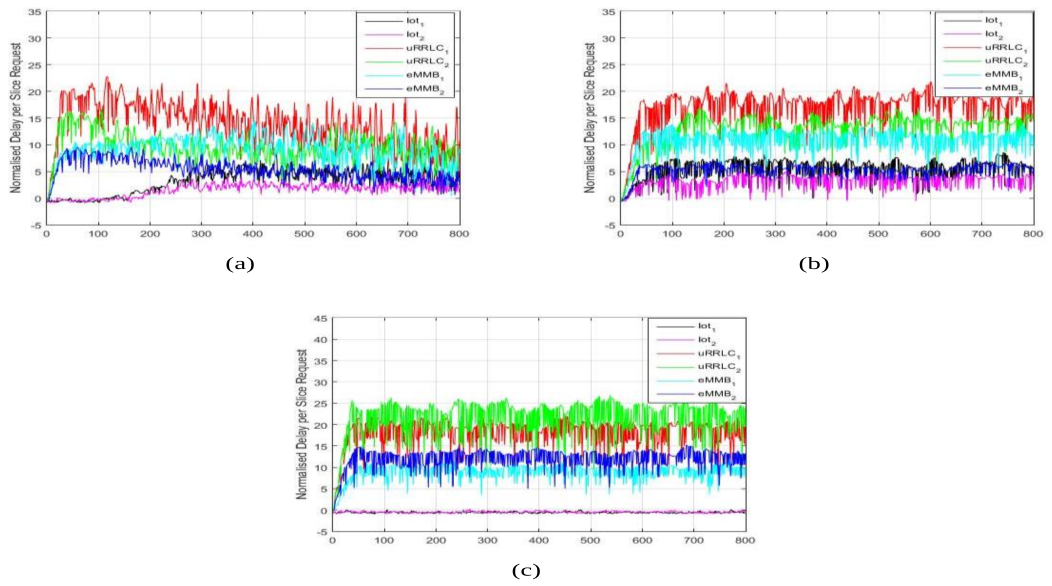


Figure 6: Normalized delays for (a) Deep Q-Learning, (b) Q-Learning and (c) Random algorithms.

5. Conclusions

In this paper, a deep reinforcement learning slice admission algorithm was designed. A utility, that incomporates rewards and the cost of slice request delays, was also designed and served as the maximisation objective for the algorithm. Results show that the Deep Q-learning slice admission algorithm has superior performance when compared to Q-learning and random slice admission algorithms; this is due to its use of historical data and the ability to extract system features, which equips it to make optimal decisions. Additionally, the results show that the utility developed enabled the algorithm maximise revenue while simultenously offering queue length and queue delay management.

6. Recommendations

In this study, the end users are assumed stationary. But in paractical mutli-tier 5G wireless networks, the users are mobile with the capability of moving from one cell to another. This movement will impact the design of slice admission algorithms.

References

[1] Alpaydin, E.: Introduction to Machine Learning/Ethem Alpaydin. The MIT Press, Cambridge (2010)

[2] N. C. Luong et al., "Applications of Deep Reinforcement Learning in Communications and Networking:

A Survey", *IEEE Commun. Surveys Tuts.*, May 2019.

[3] T. Mitchell. Machine Learning, McGraw-Hill, 1997.

[4] NGMN Alliance, "Description of Network Slicing Concept," Public Deliverable, 2016

[5] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," IEEE Commun. Mag., vol. 55, no. 5, pp. 94–100, 2017

[6] J. Ordonez-Lucena et al., "Network slicing for 5G with SDN/NFV: Concepts architectures and challenges", *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80-87, May 2017.

[7] C. Natalino et al., "Machine learning aided resource orchestration in multi-tenant networks", *IEEE SUM*, 2018..

[8] Okumu, Elizabeth M. "A Q-Learning Based Slice Admission Algorithm for Multi-Tier 5G Cellular Wireless Networks." *American Academic Scientific Research Journal for Engineering, Technology, and Sciences* 82.1 (2021): 11-18.

[9] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, ''Optimising 5G infrastructure markets: The business ofnetwork slicing,'' inProc. IEEE INFOCOM, Atlanta, GA, USA, May 2017,pp. 1–9.

[10] P. Monti, C. Natalino, M. R. Raza, P. Ohlen, and L. Wosinska, ``A slice admission policy based on reinforcement learning for a 5G flexible RAN," in Proc. Eur. Conf. Opt. Commun. (ECOC), no. 1, 2018, pp. 1_3.

[11] Bega, Dario, et al. "A machine learning approach to 5G infrastructure market optimization." *IEEE Transactions on Mobile Computing* 19.3 (2019): 498-512.

[12] Xiong, Zehui, et al. "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges." *IEEE Vehicular Technology Magazine* 14.2 (2019): 44-52.

[13] Ayodele, Taiwo Oladipupo. "Types of machine learning algorithms." *New advances in machine learning* 3 (2010): 19-48

[14] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998

[15] R. Bellman, "A markovian decision process," DTIC, Tech. Rep., 1957.

[16] E. Even-Dar and Y. Mansour, "Learning rates for Q-learning," Journal of Machine Learning Research, vol. 5, pp. 1–25, Dec. 2003.

[17] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning. MIT press Cambridge, 2016.

[18] Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network." *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017.

[19] Deep Q-Learning – Combining Neural Networks and Reinforcement Learning, Deeplizard, Accessed on November 12 2021. [Online]. Available: https://deeplizard.com/learn/video/wrBUkpiRvCA