Benchmarking Q-learning methods for intelligent network orchestration in the edge

Benchmarking Q-Learning Methods for Intelligent Network Orchestration in the Edge

Joel Reijonen*, Miljenko Opsenica*, Tero Kauppinen*, Miika Komu*, Jimmy Kjällman*, Tomas Mecklin*, Eero Hiltunen*, Jari Arkko*, Timo Simanainen*, Mohammed Elmusrati[Ψ]

*Ericsson Research, Jorvas Finland – [Ψ]University of Vaasa, Vaasa Finland

*{firstname.surname@ericsson.com}, [Ψ]mohammed.elmusrati@uva.fi

***Abstract*—We benchmark Q-learning methods, with various action selection strategies, in intelligent orchestration of the network edge. Q-learning is a reinforcement learning technique that aims to find optimal action policies by taking advantage of the experiences in the past without utilizing a model that describes the dynamics of the environment. With experiences, we refer to the observed causality between the action and the corresponding impact to the environment. In this paper, the environment for Q-learning is composed of virtualized networking resources along with their dynamics that are monitored with Spindump, an in-network latency measurement tool with support for QUIC and TCP. We optimize the orchestration of these networking resources by introducing Q-learning as part of the machine learning driven, intelligent orchestration that is applicable in the edge. Based on the benchmarking results, we identify which action selection strategies support network orchestration that provides low latency and packet loss by considering network resource allocation in the edge.**

Keywords—Q-learning, edge, intelligent orchestration

I. Introduction

Edge computing brings computing resources to the proximity of end-users and data sources. By harnessing the resources close to the data sources, edge computing can provide low latency, lower network traffic utilization in the core network, and lower data exposure to external networks due to local processing [12]. It is worth noting that the amount of available computing resources at the edge can be rather scarce compared to centralized data centers. Also, the computing and networking resources are virtualized in a modern edge network, and the virtualization enables elastic scaling and management of these resources [10]. In order to cope with the increasing number of dynamic and virtualized resources, edge orchestration systems should consider automating the management of the edge-based services and resources.

In order to improve orchestration of the network edge beyond rule-based design, machine learning is a critical factor in modern automatization. With machine learning, the orchestration system can improve its own performance without human intervention and extract hidden information from high volumes of data [1]. Moreover, orchestration systems can take advantage of machine learning to optimize operations where correct algorithms are unknown or hard to define. However, in resource constrained edge, model-based machine learning techniques can be challenging to run due to computational complexity.

In this paper, we benchmark Q-learning methods in intelligent orchestration of the network edge. Q-learning is a reinforcement-based machine learning technique that is model-free. With model-free reinforcement learning, we propose an intelligent orchestration solution that is light-weight to run even in resource constrained edge. By benchmarking Q-learning methods, with several action selection strategies, we strive to identify which action selection strategies support intelligent network edge orchestration in a way that reduces latency, packet loss, and network resource allocation.

In related studies, different reinforcement learning algorithms have been benchmarked in various use cases, such as in continuous control tasks, real-world robots and video games [7, 8, 9]. In this paper, our research contribution introduces the benchmarks of Q-learning methods as a functional part of intelligent network orchestration. In the work of D. Zeng et al. [15], Q-learning, specifically deep Q neural network, is also applied to edge resource management, but the work focuses on a virtual machine migration scenario, and it doesn't compare different Q-learning algorithms.

The rest of this paper is organized as follows. Section II introduces the background of the theoretical foundations of this paper. Section III describes the experimental setup and the test environment, while Section IV presents the benchmarking results. Finally, we conclude and describe future work in Section V.

II. Background

A. Edge

The edge can be defined as the compute and network resources on the path between data sources and clouds [6]. For example, the network edge can refer to a radio access network (RAN) part of a mobile network [12]. In modern networks, including edge networks, network functions are typically virtualized, and can be orchestrated and scaled similarly to other cloud and edge computing workloads [10, 11]. As these networks may also include physical network resources, a network controller based on the paradigm of Software Defined

Networking (SDN) could be utilized because it can support both hardware and software-based network resources.

In edge environments, the available computing resources can be served, e.g., by a small data center, and, thus, resources are often more limited than in a central data center [12]. Maintaining multiple small data centers might also be more expensive than a single large one. For these reasons, it can be argued that using an optimal amount of resources for specific flows, and freeing them to other flows when no longer needed, is of particular importance. In addition, edge environments – including the associated orchestration systems – should be able to work autonomously without a dependency to centralized clouds in order to minimize traffic into central clouds, and to operate also in the case upstream network connections or central services are unavailable [12].

B. Q-learning

Generally, in reinforcement learning, the key idea is that a reinforcement learning agent strives to maximize its expected rewards by taking actions based on the state of the environment [1]. In order to find the actions that result in maximized rewards, the agent needs to train a corresponding model or to find optimal policies that describe the correlations between the action space and rewards. In this paper, we focus on Q-learning that is a model-free reinforcement learning algorithm. By utilizing Q-learning, we achieve a light-weight solution to be executed in the edge. Compared to model-based approaches, model-free algorithms (including Q-learning), are online, require less space and are more expressive [16].

Q-learning pursues to find optimal action policies by taking advantage of the experiences without an explicit model that describes the dynamics of the environment. With experiences, we refer to the observed causality between the action and its impact on the environment. This policy finding procedure is also known as temporal-difference (TD) learning that takes advantage of model-free learning by updating estimated policies based on other learned estimations [1, 2]. In Q-learning, these estimated policies are parametrized through Q-functions, $Q(S, A)$, that identify the relative utility of selected action A in state S [4].

In [2], Q-learning has been defined in the following form:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right], \quad (1)$$

where Q is the learned action-value function, S is the state of the environment, A is the action, α is the *learning rate*, R is the gained reward, γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount factor* and t is time.

In Q-learning, and generally in reinforcement learning, the challenge is to select an action that balances the trade-off between *exploitation* and *exploration* [2]. Exploitation and exploration are concepts that describe how the agent should perform in order to achieve rewards. Exploitation refers to taking actions that have been proven to generate rewards based on previous experiences. In order to exploit actions, based on previous experiences, the agent needs to try new actions that haven't been selected before [2]. The discovery procedure of these new actions is called exploration. In this paper, we benchmark the performance of Q-learning by applying various action selection strategies.

1) Greedy action selection

Greedy action selection is a simple strategy where the agent selects an action with the highest estimated reward. This strategy utilizes a pure exploitation approach where the discovery of new actions is minimal.

We define a simple greedy action selection strategy in the following form:

$$A_i = \max_A \mathrm{Q}(S, A), \quad (2)$$

where A_i is the selected action and $\max_A \mathrm{Q}(S, A)$ is the highest Q-value for the corresponding action.

The greedy action selection strategy exploits the actions with the maximized estimated rewards in a straightforward manner. Thus, this strategy does not explore actions that may seem to produce low rewards. Due to the lack of exploration, there is a risk that the greedy selection action strategy may not ever find the most optimal actions. In fact, the greedy action selection strategy may hang onto suboptimal actions.

2) ε-greedy action selection

In ε-greedy action selection, the agent explores and selects a random action with probability ε and exploits the best-known action with probability 1- ε. With the best-known action, we refer to an action that has the highest Q-value in the given state S.

In [3], ε-greedy action selection probability has been defined in the following way:

$$P(A_i) = \begin{cases} (1-\varepsilon) + \left(\frac{\varepsilon}{n}\right), if\ \mathrm{Q}(S_t, A_t)\ is\ the\ highest \\ \frac{\varepsilon}{n}, \text{ otherwise} \end{cases}, \quad (3)$$

where $P(A_i)$ is the probability of selecting action A_i and n is the number of actions in the set.

Although greedy and ε-greedy action selection strategies favor actions that correspond to high estimated rewards, the key difference between these two strategies is in the explorative approach. In greedy action selection, the agent only exploits actions that have proven to produce high rewards in the past rather than exploring new actions. On the contrary, the agent explores new actions continuously with the probability of ε in ε-greedy action selection strategy.

3) Softmax action selection

Softmax action selection is based on softmax function that normalizes the input of real numbers into a probability distribution. In this probability distribution, the values represent probabilities that are proportional to the exponentials of the given input of real numbers.

In [4], softmax action selection with Boltzmann distribution has been defined in following form:

$$P(A_i(t)) = \frac{e^{Q_i(t)/T}}{\sum_k e^{Q_k(t)/T}}, \text{i} = 1, 2, \ldots, n., \quad (4)$$

where $P(A_i(t))$ is the probability of selecting action A_i, $Q_i(t)$ is the Q-value of the corresponding action at time t and T is the *temperature* parameter ($T > 0$) that controls the trade-off between exploitation and exploration. When $T \rightarrow 0$ the agent conducts greedy action selections by exploiting known actions that result with high rewards, and when $T \rightarrow \infty$ the agent explores new actions and their consequences [4].

The key difference between softmax and ε-greedy action selection strategies is that the agent explores new actions stochastically in ε-greedy action selection whereas, in softmax action selection, the agent ranks and weights new actions by utilizing probability distributions in exploration.

4) Deep Q neural network

In addition to algorithmic approaches, we introduce deep neural network-based Q-learning. In deep Q neural networks (DQNN), the experienced actions and the corresponding observations are used as an input when the network is trained [5]. The trained DQNN strives to approximate the corresponding Q-values for each observation rather than interpreting the Q-values based on a pre-determined algorithm. Based on the approximated Q-values, the DQNN selects the best action that maximizes the action-value function that is defined as follows [5]:

$$Q(S, A) = max_\pi E[R_t | S_t = S, A_t = A, \pi], \quad (5)$$

where π is the action selection strategy and R_t is the expected reward. In this paper, we consider Boltzmann distribution as the action selection strategy in DQNN.

III. Experimental Setup

We prototyped an intelligent network edge orchestration solution that utilizes Q-learning to optimize the dynamics of virtualized networking resources. In our solution, we introduce an intelligent agent that implements Q-learning by monitoring the environment with Spindump to optimize the orchestration of the edge domain (Figure 1). Spindump is an open-source tool for in-network latency measurements with support for QUIC and TCP [13].

In the proposed network edge orchestration solution, the orchestrator automatically scales virtual networking resources and reconfigures the network between clients and the application server based on the monitored information on latency and packet loss. With the assistance of the intelligent agent, the orchestrator aims to provide a high quality of the service to the clients by minimizing the network latency and packet loss.

The intelligent agent monitors the data from Spindump and applies Q-learning for orchestration in real-time. In Q-learning, the agent can utilize various action selection strategies, such as greedy, ε-greedy, softmax and deep Q neural network strategies in order to determine the actions that would result in the highest expected rewards in orchestration. These actions refer to scaling of virtualized networking resources, and the rewards are directly proportional to the fulfilment of the quality requirements. The action space of the agent is composed of the following actions: scale out a resource, scale in a resource or do nothing. If the agent enforces orchestration actions that decrease latency, packet loss and resource allocation, then, within the edge domain, the agent receives high rewards.

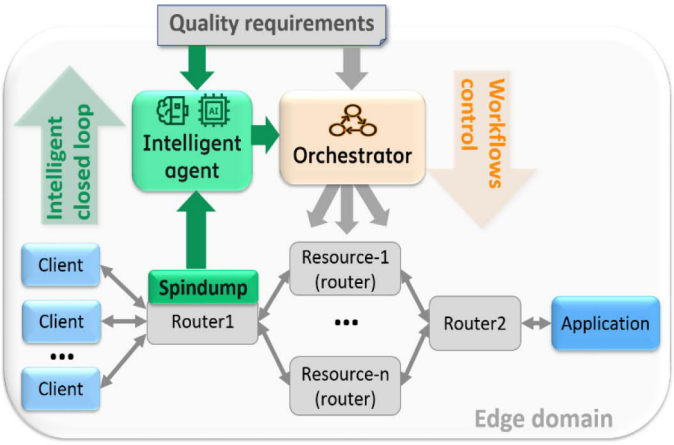

Fig. 1. Overview of the intelligent network orchestration solution

In our solution, the intelligent agent considers following parameters in Q-learning:

- Action space, $A = [a_1, a_2, a_3]$, where a_1 is an action that scales out a resource, a_2 is an action that scales in a resource and a_3 is an action that does nothing.
- State space, S = [max(c) × max(nr)], where max(c) is the maximum number of clients and max(nr) is the maximum number of networking resources that can be allocated in the edge domain.
- Reward space, $R = [r_1, r_2, r_3, r_4]$, where r_1 is a reward when quality requirements are completely fulfilled, r_2 is a reward when only latency requirements are fulfilled, r_3 is a reward when only packet loss requirements are fulfilled and r_4 is a reward when none of the quality requirements are fulfilled. Furthermore, rewards are calculated as follows: $r_1 = 5 + n_c / n_{nr}$, $r_2 = -5 * L_m / L_r$, $r_3 = -5 * P_m / P_r$ and $r_4 = -5 * L_m / L_r + -5 * P_m / P_r$ where n_c is the number of clients, n_{nr} is the number of utilized networking resources, L_m is the monitored latency, L_r is the latency requirement, P_m is the monitored packet loss and P_r is the packet loss requirement.
- Learning rate, $\alpha = 0.15$
- Discount factor, $\gamma = 0.6$
- Epsilon, $\varepsilon = 0.05$
- Temperature parameter, $T = 1$

Additionally in DQNN, the agent considers following neural network structure which is implemented using Keras [17]:

- Input layer size: 4

- Output layer size: 3
- Number of hidden layers: 3 (16-16-16)
- Number of epochs: 500
- Batch size: 32
- Optimizer: Adam

In order to simulate the network edge, our experimental setup is implemented in a single OpenStack based virtual machine (VM) hosted in Ericsson Research Data Center in Lund. The VM has a relatively constrained environment with only 2 virtual processors and 4 GB of memory. However, the resource allocation is sufficient because the entire testbed consists of a limited number of virtualized resources that generate a low amount of computational load and traffic.

In the experimental setup, clients generate request-reply traffic with ICMP at a regular interval towards a common server. The emulated networking between the clients and the server is implemented with Docker networking features [14]. The emulated network consists of two virtualized routers, which are not scaled in this setup, are connected to a scalable pool of virtualized networking resources that route packets within the network. The routers utilize equal-cost multipath routing (ECMP) to load balance traffic across resource instances. The network resources can handle a fixed amount of networking traffic where the traffic is regulated with traffic control (tc). Additionally, the resources use queueing discipline (qdisc) to limit the bandwidth, and also to add latency to the link.

IV. Results and Discussion

We benchmarked Q-learning methods by applying various action selection strategies for intelligent network orchestration in the simulated network edge. We implemented an agent that introduces Q-learning as a part of intelligent orchestration solution. During benchmarking, our target was to obtain latency of less than 90 ms and a packet loss of less than 0.5% by using minimal amount of networking resources.

We benchmarked the performance of a heuristic, a greedy action selection strategy, an ε-greedy action selection strategy, a softmax action selection strategy, and a deep Q neural network (DQNN) approaches. With the heuristic approach, we refer to a simple orchestration logic where we have a single resource unit serving 2.5 clients. This ratio is defined by calculating how many clients a single resource unit can theoretically serve. Other strategies are Q-learning based where the agent aims to learn the optimal ratio of resources and clients via exploration and exploitation. We benchmarked Q-learning methods after each method had converged.

Our benchmarking procedure consists of cyclic increases and decreases of clients where the number of clients varies between 1 and 15 and the maximum number of networking resources is 10. Each client produces request-response traffic with no congestion control. Furthermore, we calculated both the mean and the standard deviation of each benchmark over the benchmarking cycle that lasted for 12 hours.

In a comparison of latency benchmarks, each approach fulfills our target of having latency of less than 90 ms (Figure 2). In this benchmark, ε-greedy action selection strategy achieves the lowest latency whereas the heuristic approach introduces the highest latency.

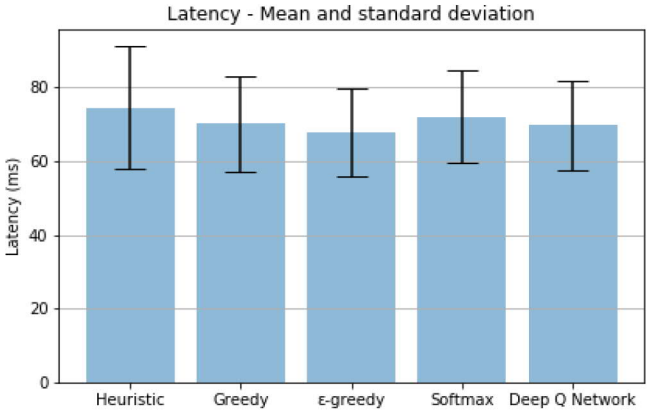

Fig. 2. Comparison of latency by applying benchmarked action selection strategies

In a comparison of packet loss, both heuristic approach and greedy action selection strategy fail to fulfill our target of having packet loss of less than 0.5%, whereas others succeed (Figure 3). DQNN achieves a 'perfect' result by achieving zero packet loss during the benchmarking cycle of 12 hours. The main difference between the DQNN and the other action selection strategies is that the DQNN approximates the Q-values for each observation whereas the other strategies interpret the Q-values based on a pre-defined algorithm. Due to the dynamics of the virtualized resources in the experimental setup, approximations appear to be more efficient than algorithmic interpretations when it comes to the optimization of the packet loss.

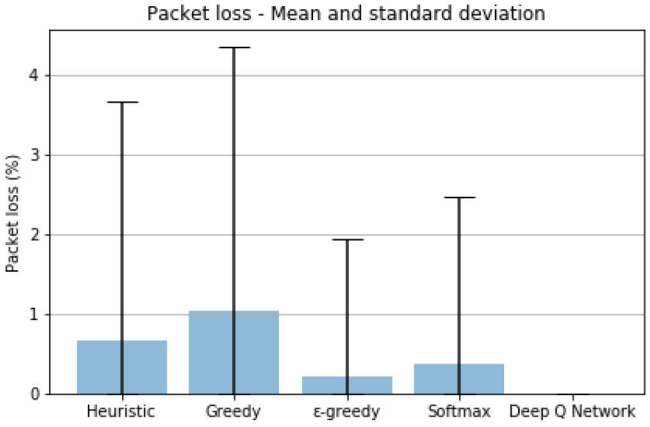

Fig. 3. Comparison of packet loss by applying benchmarked action selection strategies

In a comparison of network resource allocation, DQNN allocates significantly more resources compared to the other approaches (Figure 4). Therefore, the 'perfect' result in packet loss can be explained by the increased utilization of the resources. Other approaches than DQNN allocate a bit less than four resource instances, except the greedy action selection strategy that achieves the minimum in resource allocation.

However, the greedy strategy failed to fulfill the initial performance target by resulting packet loss of more than 1%.

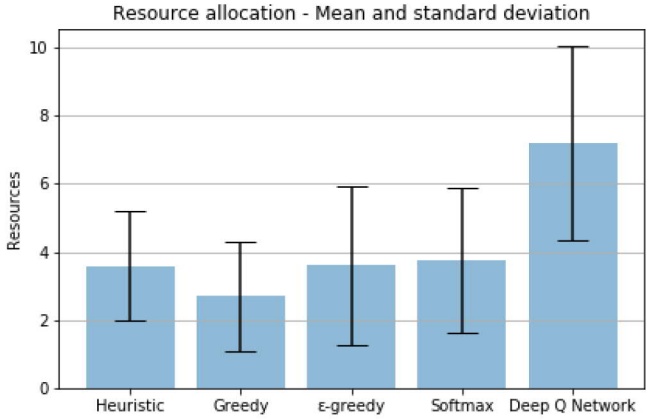

Fig. 4. Comparison of network resource allocation by applying benchmarked action selection strategies

Based on benchmarking results, ε-greedy seems to be the best action selection strategy to be used for resource allocation in the simulated network edge since it succeeds to fulfill the targets of having low latency and packet loss, and also it allocates a relatively low number of resources. Alternatively, softmax would be another action selection strategy that would be feasible to be deployed for resource allocation in the edge. DQNN achieves the 'perfect' result by achieving zero packet loss but, on the other hand, the neural network allocates significantly more resources in orchestration, and thus, it would not necessarily be preferred in the resource-constrained edge.

V. Conclusions and Future Work

This paper provides benchmarking results of Q-learning methods by applying various action selection strategies in intelligent network orchestration in the edge. We applied greedy, ε-greedy, softmax and deep Q neural network (DQNN) action selection strategies in a prototype of an intelligent network edge orchestration solution. In the prototyped solution, the orchestrator automatically scales virtualized networking resources and reconfigures the network according to the volume of clients by taking advantage of Q-learning. Based on the benchmarking results, ε-greedy and softmax action selection strategies are feasible solutions to be used in the edge since they fulfill our defined network performance targets by introducing low latency and packet loss. Additionally, these strategies allocate a moderate amount of network resources whereas DQNN allocates significantly more resources compared to the other strategies. However, DQNN is the only approach that achieves zero packet loss which indicates that approximation-based Q-value definition is efficient when it comes to the optimization of complex dynamics.

Intelligent orchestration plays a significant role in next generation telecommunication technologies and networks, such as in 5G and beyond. In advanced telecommunication networks, the network edge is becoming more flexible, enabling more heterogeneous traffic models applicable to an expanding set of use cases, thus introducing multiple research challenges. For instance, the complexity of the operations should not avert these edge-based solutions from being deployed in large scale. Overlay networking and end-to-end encryption bring yet additional layers of complexity into network optimization. In order to orchestrate the advanced telecommunication networks efficiently, the described features should be considered in the design of machine learning driven orchestration.

References

[1] E. Alpaydin, Introduction to machine learning, 2010, pp. 3-12, 30-34, 447-455

[2] R. S. Sutton & A. G. Barto, Reinforcement learning: an introduction, 2014, pp. 32-44, 95-98

[3] E. R. Gomes & R. Kowalczyk, Dynamic analysis of multiagent Q-learning with ε-greedy exploration, 2009, Proceedings of the 26th international conference on machine learning, pp. 369-376

[4] A. Kianercy & A. Galstyan, Dynamics of Boltzmann Q-learning in two-player two-action games, 2011, arXiv, [online], available: https://arxiv.org/abs/1109.1528

[5] T. Oda, R. Obukata, M. Ikeda, L. Barolli & M. Takizawa, Design and implementation of a simulation system based on deep Q-network for mobile actor node control in wireless sensor and actor networks, 2017, 31st international Conference on advanced Information networking and applications workshops (WAINA), pp. 195-200

[6] W. Shi, J. Cao, Q. Zhang, Y. Li & L. Xu, Edge computing: vision and challenges, 2016, IEEE Internet of Things Journal, vol 3, no. 5, pp. 637–646

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman & P. Abbeel, Benchmarking Deep Reinforcement Learning for Continuous Control, 2016, Proceedings of the 33rd International Conference on Machine Learning

[8] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma & J. Bergstra, Benchmarking reinforcement learning algorithms on real-world robots, 2018, 2nd Conference on Robot Learning

[9] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel & J. Ba, Benchmarking model-based reinforcement learning, 2019, Preprint, arXiv, [online], available: https://arxiv.org/pdf/1907.02057.pdf

[10] O. Adamuz-Hinojosa, J. Ordonez-Lucena, P. Ameigeiras, J. J. Ramos-Munoz, D. Lopez & J. Folgueira, Automated network service scaling in NFV: concepts, mechanisms and scaling workflow, 2018, in IEEE Communications Magazine, vol. 56, no. 7, pp. 162-169

[11] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta & D. Sabella, On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration, 2017, in IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1657-1681

[12] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong & J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, 2019, Journal of Systems Architecture, Volume 98, pp. 289-330

[13] J. Arkko, Gearing-up for modern Internet transport, 2019, Ericsson research blog, [online], available: https://www.ericsson.com/en/blog/2019/1/quic

[14] Docker documentation, 2019, [online], available: https://docs.docker.com/network/

[15] D. Zeng, L. Gu, S. Pan, J. Cai & S. Guo, Resource Management at the Network Edge: A Deep Reinforcement Learning Approach, 2019, in IEEE Network, vol. 33, no. 3, pp. 26-33

[16] C. Jin, Z. Allen-Zhu, S. Bubeck & M. I. Jordan, Is Q-learning provably efficient? 2018, arXiv, [online], available: https://arxiv.org/abs/1807.03765

[17] Keras: The Python Deep Learning library, 2020, [online], available: https://keras.io