IEICE TRANS. INF. & SYST., VOL.E106–D, NO.11 NOVEMBER 2023

PAPER

A Lightweight Reinforcement Learning Based Packet Routing Method Using Online Sequential Learning

Kenji NEMOTO[†a)], *Nonmember and* **Hiroki MATSUTANI**[†b)], *Member*

SUMMARY Existing simple routing protocols (e.g., OSPF, RIP) have some disadvantages of being inflexible and prone to congestion due to the concentration of packets on particular routers. To address these issues, packet routing methods using machine learning have been proposed recently. Compared to these algorithms, machine learning based methods can choose a routing path intelligently by learning efficient routes. However, machine learning based methods have a disadvantage of training time overhead. We thus focus on a lightweight machine learning algorithm, OS-ELM (Online Sequential Extreme Learning Machine), to reduce the training time. Although previous work on reinforcement learning using OS-ELM exists, it has a problem of low learning accuracy. In this paper, we propose OS-ELM QN (Q-Network) with a prioritized experience replay buffer to improve the learning performance. It is compared to a deep reinforcement learning based packet routing method using a network simulator. Experimental results show that introducing the experience replay buffer improves the learning performance. OS-ELM QN achieves a 2.33 times speedup than a DQN (Deep Q-Network) in terms of learning speed. Regarding the packet transfer latency, OS-ELM QN is comparable or slightly inferior to the DQN while they are better than OSPF in most cases since they can distribute congestions.

key words: *reinforcement learning, packet routing, neural networks, OS-ELM*

1. Introduction

In the past few years, the amount of traffic flowing through the Internet has increased rapidly [1]. Existing routing protocols such as OSPF [2] and RIP [3] may not be able to deal with the increase of network traffic. For example, OSPF protocol uses Dijkstra algorithm to find the shortest path without considering the congestion. Therefore, when a data flow increases, it can overload certain routers and reduce a throughput in the network. On the other hand, packet routing methods using machine learning have been proposed recently [4], [5]. These methods can intelligently select a routing path by utilizing a high representational ability to take into account complex information. Some previous works report that machine learning based methods achieve a higher throughput than OSPF [4], [6].

However, many of the previous methods only aim to improve a packet transfer efficiency, and there has been little research on reducing the training costs. A lower training cost has some advantages. Since network conditions change from time to time, it is better to shorten the training time to deal with such changes. In addition, all the network nodes may not have computing resources enough to train deep neural networks. In this paper, we propose a packet routing method using OS-ELM (Online Sequential Extreme Learning Machine) [7], which enables a sequential learning of neural networks. It is known as a lightweight machine learning method compared to deep neural networks using a backpropagation algorithm.

A reinforcement learning method using OS-ELM for Q-learning has already been proposed [8]. This previous work used a random update technique, which has a disadvantage of slow convergence speed during the training. In this paper, we newly introduce an experience replay buffer to OS-ELM based reinforcement learning in order to stabilize the training. In addition, we design the state and reward to achieve a lower latency in the packet transfer. Specifically, we aim to avoid congestions by using negative numbers of delays in the reward. We evaluate OS-ELM QN as a reinforcement learning based packet routing method.*

The rest of this paper is organized as follows. Section 2 describes preliminary knowledge. Section 3 overviews related works. Section 4 proposes OS-ELM QN for packet routing. Evaluation results in terms of learning performance, execution time, and packet routing performance are presented in Sect. 5. Section 6 discusses the evaluation results of OS-ELM QN. Section 7 concludes this paper.

2. Preliminaries

This section introduces OS-ELM, DQN (Deep Q-Network), and prioritized replay buffer.

2.1 OS-ELM

OS-ELM [7] is an online sequential learning algorithm for 3-layer neural networks that consist of an input layer, a hidden layer, and an output layer. Here, we assume that the numbers of their nodes are n, $\tilde{X}$, and m, respectively. Figure 1 shows an example network model of OS-ELM.

In this figure, $\boldsymbol{\alpha} \in \boldsymbol{R}^{n \times \tilde{X}}$ is an input weight matrix between the input and hidden layers, $\boldsymbol{\beta} \in \boldsymbol{R}^{\tilde{X} \times m}$ is an output weight matrix between the hidden and output layers, and $\boldsymbol{b} \in \boldsymbol{R}^{\tilde{X}}$ is a bias vector of the hidden layer.

Manuscript received December 28, 2022.
Manuscript revised May 1, 2023.
Manuscript publicized August 15, 2023.

[†]The authors are with Graduate School of Science and Technology, Keio University, Yokohama-shi, 223–8522 Japan.
a) E-mail: kenji@arc.ics.keio.ac.jp
b) E-mail: matutani@arc.ics.keio.ac.jp

DOI: 10.1587/transinf.2022EDP7231

*This paper is an extended version of [9] by adding evaluation results of the scalability.

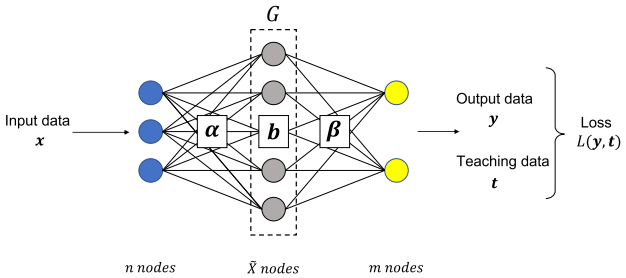

Fig. 1 OS-ELM (Online Sequential Extreme Learning Machine)

Assuming that the i-th input data $\boldsymbol{x}_i \in \boldsymbol{R}^{k\times n}$ and teacher label $\boldsymbol{t}_i \in \boldsymbol{R}^{k\times m}$ with batch size k are given, the i-th optimal solution $\boldsymbol{\beta}_i$ can be computed as the following equation.

$$\begin{aligned}\boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \boldsymbol{P}_{i-1}\boldsymbol{H}_i^\top(\boldsymbol{I} + \boldsymbol{H}_i\boldsymbol{P}_{i-1}\boldsymbol{H}_i^\top)^{-1}\boldsymbol{H}_i\boldsymbol{P}_{i-1}\\ \boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i\boldsymbol{H}_i^\top(\boldsymbol{t}_i - \boldsymbol{H}_i\boldsymbol{\beta}_{i-1}),\end{aligned} \tag{1}$$

where $\boldsymbol{H}_i$ is defined as $\boldsymbol{H}_i \equiv G(\boldsymbol{x}_i \cdot \boldsymbol{\alpha} + \boldsymbol{b})$ using an activation function G.

The initial values $\boldsymbol{P}_0$ and $\boldsymbol{\beta}_0$ are precomputed as follows.

$$\begin{aligned}\boldsymbol{P}_0 &= (\boldsymbol{H}_0^\top\boldsymbol{H}_0)^{-1}\\ \boldsymbol{\beta}_0 &= \boldsymbol{P}_0\boldsymbol{H}_0^\top\boldsymbol{t}_0\end{aligned} \tag{2}$$

As shown in Eq. (1), the output weight matrix $\boldsymbol{\beta}_i$ and its intermediate result $\boldsymbol{P}_i$ are computed from the previous training results $\boldsymbol{\beta}_{i-1}$ and $\boldsymbol{P}_{i-1}$. Thus, OS-ELM can sequentially update the model with a newly-arrived target chunk in one shot.

2.2 DQN

DQN [10] is known as a typical reinforcement learning algorithm. $Q_{\theta_1}(s_t, a_t)$ represents a Q-value in time step t when taking action a_t in state s_t. θ_1 represents a set of neural network parameters.

In DQN, a target signal can be computed as follows.

$$f(r_t, s_{t+1}, d_t) = r_t + (1 - d_t)\gamma \max_{a\in A} Q_{\theta_2}(s_{t+1}, a), \tag{3}$$

where $\gamma \in [0, 1]$ is a discount rate that determines the importance of the next step, r_t represents the reward for transitioning from s_t to s_{t+1}, and d_t indicates whether the episode is finished, which is expressed as 1 or 0. In addition, Eq. (3) uses a fixed target Q-Network technique. If θ_1 is changed each time while it is used for predicting the Q-value, the training process becomes unstable. To mitigate this issue, a target Q-Network θ_2 is separated from the main Q-Network θ_1. θ_2 is used to generate the target Q-value for the reinforcement learning while it is periodically updated by θ_1. Then, the loss value $L(\theta_1)$ is computed with the following equation.

$$L(\theta_1) = \mathbb{E}[(Q_{\theta_1}(s_t, a_t) - f(r_t, s_{t+1}, d_t))^2], \tag{4}$$

where $\mathbb{E}$ means an expectation value. The current Q-Network parameters θ_1 are updated to minimize the squared error loss between the main Q-Network and the target Q-Network as shown in the above equation.

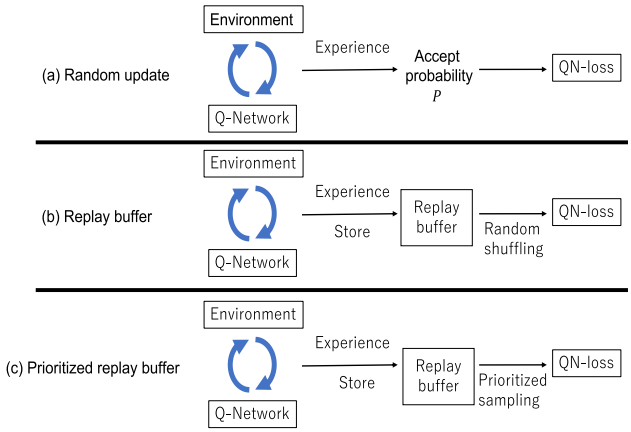

Fig. 2 Three experience sampling techniques

In DQN, an experience replay technique [10] is used to acquire a set of experiences. In Fig. 2 (b), a replay buffer uses this technique. An experience means a set of s_t, a_t, r_t, s_{t+1}, and d_t. If a training is performed every time an experience is generated, the training is affected by a temporal dependence of the generated experiences. To mitigate this negative impact, a batch of experiences is randomly picked up from the replay buffer.

2.3 Prioritized Replay Buffer

The random experience sampling method mentioned in Sect. 2.2 is sometimes inefficient to train certain transitions of high importance. To address this issue, a recent reinforcement learning uses a prioritized experience replay buffer technique [11].

The prioritized experience replay buffer technique assigns a weight to the sampling probability of each experience based on a priority. The priority can be calculated by the TD (Temporal Difference) error [11]. In Fig. 2 (c), a replay buffer uses this prioritized sampling. A sampling probability of an experience i is calculated based on priority p of the experience as follows.

$$P_i = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \qquad (p_k \neq 0), \tag{5}$$

where α is a hyper-parameter that weights the priority; if α is 0, it is equivalent to a random sampling.

3. Related Work

3.1 Reinforcement Learning Using OS-ELM

As mentioned earlier, an OS-ELM based reinforcement learning is proposed [8]. This section introduces its related techniques and their shortcomings that degrade the learning efficiency.

3.1.1 Simplified Output Model

The loss value of DQN is calculated by Eq. (4). In typical DQNs, the i-th node of an output layer represents the Q-value of the i-th action, and the Q-Network is trained so that the i-th node can predict $Q(s, a_i)$. On the other hand, as Eq. (1) shows, OS-ELM requires teacher data $\boldsymbol{t} \in \boldsymbol{R}^m$ to update $\boldsymbol{\beta}$. Since OS-ELM analytically derives the weight parameters, it is necessary to specify inputs and outputs explicitly. In the previous work [8], a set of state variables and a scalar variable that represents actions is given as an input to the neural network. However, we consider that representing an action as a scalar value may not be scalable, and thus we use a vector variable instead of a scalar variable. We will describe the detail in Sect. 4.1.1.

3.1.2 Random Update

As mentioned in Sect. 2.2, DQNs typically train their neural network parameters in a batch manner and use the experience replay techniques to form a batch of sampled experiences. On the other hand, the previous work [8] fixes the batch size to 1 and randomly decides whether to train an incoming experience. Figure 2 (a) illustrates the random update technique, in which Q-Network accepts the experiences by probability P.

By fixing the batch size to 1, an inverse matrix operation $(\boldsymbol{I} + \boldsymbol{H}_i \boldsymbol{P}_{i-1} \boldsymbol{H}_i^\top)^{-1}$ in Eq. (1) can be interpreted as a simple division $\frac{1}{1+(\boldsymbol{H}_i \boldsymbol{P}_{i-1} \boldsymbol{H}_i^\top)}$ when training an incoming experience. This technique can eliminate the need for the pseudo-inverse operation that typically requires SVD or QR decomposition. However, the training may be affected by the temporal dependence since a replay buffer is not used. In this paper, we compare this technique with other sampling techniques.

3.1.3 Forget Rate

The distribution of data given by the environment may change as time goes by. To gradually reduce the impact of old trained data, $\boldsymbol{P}_{i-1}$ is recomputed before updating the parameters using Eq. (1) as follows.

$$\boldsymbol{P}_{i-1} \leftarrow \boldsymbol{P}_{i-1} / \lambda^2, \tag{6}$$

where $\lambda \in [0, 1]$ is a forget rate. Thus, the impact of old trained data is adjusted by the λ parameter.

3.2 Packet Routing Using Machine Learning

This section introduces packet routing methods based on supervised learning and reinforcement learning.

Supervised learning methods: In [4], a supervised learning method is proposed to optimize a routing efficiency using three phases: Initial phase, Learning phase, and Running phase. In Initial phase and Learning phase, each node collects traffic information by using existing routing methods (e.g., OSPF) and trains neural network models. In Running phase, each node routes packets by using the trained neural networks. It achieves a higher forwarding efficiency than OSPF by reducing signaling overhead [4].

In [12], a data-driven supervised learning model is designed to learn an efficient packet routing strategy based on given traffic demands. It implies that predicting traffic conditions accurately would be challenging.

Deep reinforcement learning methods: As mentioned in [5], DRL (Deep Reinforcement Learning) methods are superior to supervised methods in some respects. For example, supervised training methods require labeling a large amount of information in the network, which is an arduous task. Since DRL agent continuously observes states, executes actions, and receives rewards by interacting with an environment, it can cope with changes of the network conditions.

In [6], a system model of the reinforcement learning (i.e., states, actions, and rewards) is defined and a central controller which routes all the packets is designed. It shows a higher forwarding efficiency than OSPF.

4. Proposed Routing Method

In this section, we propose OS-ELM QN (OS-ELM Q-Network), which is an improved version of the reinforcement learning method using OS-ELM [8].

4.1 OS-ELM Q-Network

The agent and learning algorithms are shown in Algorithms 1 and 2.

4.1.1 Agent Algorithm

An agent generates experiences by interacting with a given environment. Algorithm 1 shows its algorithm. The agent is located in an environment and has a neural network (lines

Algorithm 1 Agent

```
1:  procedure Agent(B, T)
2:      θ1 ← initial parameters
3:      s0 ← initial state of environment
4:      for t = 1 ... T do
5:          if random value r̂ < (1 − ε) then
6:              for i = 1 ... |A| do
7:                  Add Q(s_{t−1}, â_i) into ActionList
8:              end for
9:              a_{t−1} ← arg max(ActionList)
10:         else
11:             a_{t−1} ← random action
12:         Take a_{t−1} and acquire (s_t, r_t, d_t) from environment
13:         Add τ (s_{t−1}, a_{t−1}, s_t, r_t, d_t) and priority p into a local buffer
14:         if size of local buffer > B then
15:             Update θ1 by Learning algorithm
16:         Copy weights θ1 into target network θ2 periodically
17:     end for
```

Algorithm 2 Learning

```
1:  procedure Learning
2:      τ, id ← experiences and their indexes sampled from local buffer
3:      Update priorities of experiences in local buffer using id
4:      t̂ ← r_t + (1 − d_t)γ max_{a∈A} Q_{θ2}(s_t)
5:      if initial learning then
6:          P_0 ← (H_0^⊤ H_0)^{-1}
7:          β_0 ← P_0 H_0^⊤ t̂_0
8:          Update OS-ELM QN with parameter β_0
9:          i ← 0
10:     else
11:         P_{i-1} ← P_{i-1}/λ^2
12:         P_i ← P_{i-1} − P_{i-1} H_i^⊤ (I + H_i P_{i-1} H_i^⊤)^{-1} H_i P_{i-1}
13:         β_i ← β_{i-1} + P_i H_i^⊤ (t̂ − H_i β_{i-1})
14:         Update OS-ELM QN with parameter β_i
15:     i ← i + 1
```

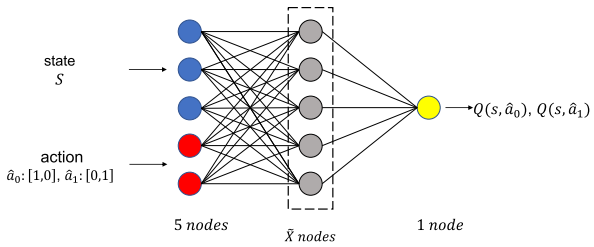

Fig. 3 OS-ELM QN model

2-3). It chooses an action according to the ϵ-greedy policy (line 5) [10]. That is, an action is chosen randomly with probability ϵ or using an inferred result with probability (1-ϵ).

The action is represented as a vector (lines 6-9). Compared to the previous method [8], we change the input format. As we described in Sect. 3.1.1, a set of state variables and a scalar action variable is used as an input data for the Q-Network [8]. However, if a single scalar is used to represent multiple actions, different scalar values are defined for different actions (e.g., 0.5 for a_0 and -0.5 for action a_1). The mapping between the scalar values and corresponding actions may affect the results. In this paper, the action is given as a vector instead of a scalar. For example, assuming that the number of input states is 3 and the number of input actions is 2. In this case, the neural network model is designed as shown in Fig. 3. The action vector is fed to this single neural network as $[1, 0]$ for action $\hat{a}_0$ and $[0, 1]$ for action $\hat{a}_1$, in addition to the state variables. The output of the neural network is the Q-value corresponding to the given action and state. Using this neural network, we get two Q-values, $Q(s, \hat{a}_0)$ and $Q(s, \hat{a}_1)$, by inferring twice with different action vectors. The action that outputs the largest Q-value is selected as the next action.

Then, the agent takes the selected action and acquires the next state s_t, reward r_t, and finish flag d_t (line 12). The agent stores the experience in a local buffer and calculates its priority (line 13). When a certain amount of experiences B are stored in the local buffer, the learning algorithm (see Algorithm 2) is executed (line 15). The target Q-Network parameters θ_2 are periodically updated by parameters θ_1 (line 16).

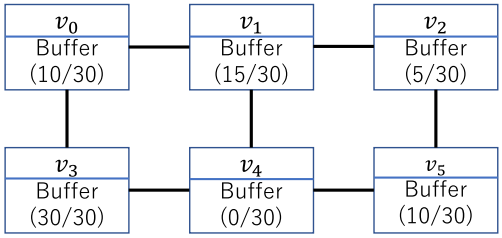

Fig. 4 Network example of six nodes. Values under "Buffer" represent (occupied capacity / total capacity).

4.1.2 Learning Algorithm

Algorithm 2 shows an algorithm that updates the Q-Network parameters. Experiences are sampled from the local buffer (line 2). It updates the priorities of the sampled experiences in the buffer based on the TD error (line 3).

As we described in Sect. 3.1.2, a random update technique is used in the previous work [8]. Compared to the previous work, we use a prioritized experience replay. This is because the previous work [8] targets low-end edge devices with a strict memory size limitation. However, in this paper, our targets are network routers which are expected to have more memory capacity than tiny edge devices. Therefore, we introduce a prioritized experience replay buffer into OS-ELM QN. This modification is expected to improve the learning performance.

The OS-ELM QN parameter β is updated by using OS-ELM algorithm that we described in Sect. 2.1. If the Q-Network is trained at the first time, an initial learning is executed (lines 5-9). After that, a sequential learning is executed to train the Q-Network (lines 11-14).

4.2 Packet Routing Using Reinforcement Learning

This section proposes a packet routing method using the reinforcement learning method.

4.2.1 System Model

The proposed method considers the following model. Here, we assume that the network is an undirected graph $G = \{\boldsymbol{V}, \boldsymbol{E}\}$, and N is the number of nodes in the network. Each node has a Q-Network for each destination node. For example, as shown in Fig. 4, node v_0 has five Q-Networks, Q_{v_0,v_1}, Q_{v_0,v_2}, Q_{v_0,v_3}, Q_{v_0,v_4}, and Q_{v_0,v_5}. Below, we describe the design of the state, action, and reward in our reinforcement learning system. The state s is an N-dimensional vector that ranks the network nodes based on their buffer occupancies. For example, there are six nodes in Fig. 4. The number of packets held by these six nodes are 10, 15, 5, 30, 0, and 10 packets, respectively. They are then sorted by their buffer occupancies in descending order. Their ranks are 3rd, 5th, 2nd, 6th, 1st, and 4th, respectively. Their ranks are used as

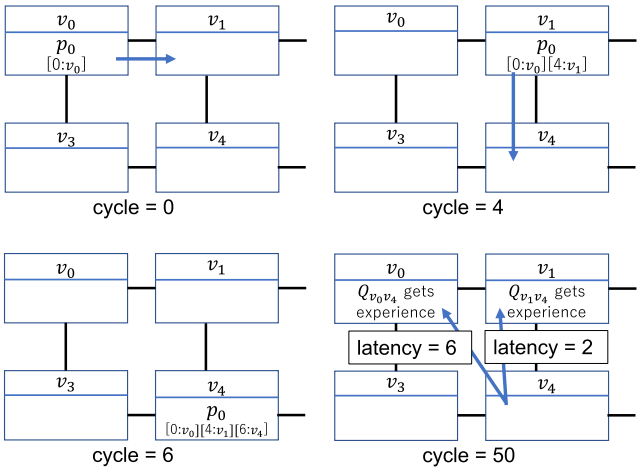

Fig. 5 Routing example. $p_0[x{:}v_i]$ means p_0 reached v_i at x cycle.

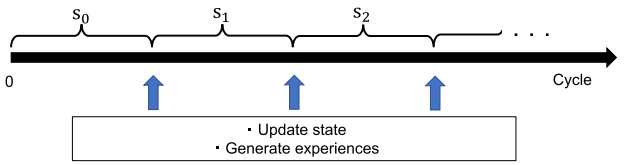

Fig. 6 Time steps for learning. State and experiences are updated at pre-specified time interval.

input for OS-ELM QN; in this case, the input ranked vector is $[3, 5, 2, 6, 1, 4]$.

The reason for using the ranked data is to stabilize the training. This is based on prior research in which the training was unstable when the actual packet counts were used as input data. In this paper, we assume current network states are propagated over the network as well as conventional practical routing protocols, such as OSPF. The state is updated periodically as we describe below.

The action space A of each Q-Network is defined as the next hop nodes that can take the shortest path for a given destination node. For example, in Fig. 4, the action space of Q_{v_0,v_5} is $[v_1, v_3]$ because node v_1 or v_3 can be selected as the next hop when node v_0 sends packets to node v_5.

The reward r is the negative signed latency of each packet to the destination node. Figure 5 shows a packet routing example. Here, we assume that a packet is sent from node v_0 to node v_4. Q_{v_0,v_4} forwards the packet p_0 from node v_0 to node v_1 at cycle 0. Then, node v_1 receives p_0 at cycle 4, and Q_{v_1,v_4} forwards p_0 to node v_4 at the same cycle. At last, node v_4 receives p_0 at cycle 6. These latencies and a routing path are stored in the packet header of p_0. As a result, Q_{v_0,v_4} will get the reward -6, and Q_{v_1,v_4} will get the reward -2. The method to acquire the rewards is described in Sect. 4.2.2.

4.2.2 Time Steps for Learning

Figure 6 shows a timing diagram of the learning process in this paper. The state is updated and experiences are generated at a pre-specified time interval.

Updating state: The state s is updated periodically. For example, assume that the update interval is 50 cycles. In Fig. 6, the state s_0 is updated with the latest ranked buffer occupancies at 0 cycle. Then, every Q-Network in every node uses the same state s_0 to select the next hop router during 0 to 49 cycles. After that, the state is updated to s_1 with the latest buffer occupancies at 50 cycle, and every Q-Network uses the same state s_1 during 50 to 99 cycles. The above steps are repeated until one episode is completed.

Generating experiences: The latencies to reach intermediate hops are recorded in each packet during its flight to the destination. In Fig. 5, p_0 has two latency values when it arrives at node v_4 from node v_0. That is, Q_{v_0,v_4} took 6 cycles to forward p_0 and Q_{v_1,v_4} took 2 cycles under s_0. At the same time as the states are updated, experiences are sent to the nodes along the routing path. In this case, Q_{v_0,v_4} receives an experience $\{s_0, v_1, -6, s_1, 0\}$ and Q_{v_1,v_4} receives an experience $\{s_0, v_4, -2, s_1, 0\}$ at 50 cycle.

5. Evaluations

5.1 Simulation Environment

All the programs are executed on a computer with an Intel Core i7-10700 CPU, 32GB RAM, and NVIDIA GTX 3060 Ti GPU. The operating system is Ubuntu 20.04.

We build a network simulator with Python 3.8.10. We use the Python libraries of Gym 0.18.3 [13] and NetworkX 2.6.3 [14] to build the environment. To build neural network models, we use Tensorflow 2.12.0 and NumPy 1.23.1.

The network topologies examined in this paper are 4x4 mesh, 8x8 mesh networks, and ARPANET. We assume a 4x4 mesh network unless otherwise stated. All the nodes and edges in a network have the same link bandwidth. Each node can forward a single packet in each cycle under a round-robin scheduling; thus 16 and 64 packets are forwarded in total in the 4x4 mesh and 8x8 mesh networks, respectively. Each node has an infinite FIFO (First-In First-Out) queue as a packet buffer. The latency for the packet transfer is 2 cycles per a hop. All the generated packets have the same size. A certain amount of packets are injected at randomly-selected (discrete uniform distribution) nodes each cycle. One episode is finished when all the packets have been delivered completely. A destination node of each packet is also selected randomly. The update interval we described in Sect. 4.2.2 is set to 50 cycles. The update interval of states is a tuning parameter. We evaluate rewards of OS-ELM QN by varying the update interval from 10 to 150 cycles. The hyperparameters are listed in Table 2. A 4x4 mesh topology is used. The model uses a replay buffer. The reward value is a sum of those in all the nodes. Figure 7 shows the evaluation result, where the rewards are averaged over ten trials. The light-colored lines represent the average values, while the heavy-colored lines represent moving averages of five values. From this evaluation result, we observe that the update intervals longer than 50 cycles decrease the total rewards. On the other hand, shorter update intervals

Table 1 Five routing methods compared in this paper.

	Algorithm	Number of layers	Sampling method
(1)	DQN	4	Replay buffer
(2)	DQN	4	Prioritized replay buffer
(3)	OS-ELM QN	3	Random update
(4)	OS-ELM QN	3	Replay buffer
(5)	OS-ELM QN	3	Prioritized replay buffer

Table 2 Hyperparameters of DQN and OS-ELM QN.

Batch size	64
ϵ-greedy ϵ	0.05
Forget rate λ	0.99
Discount rate γ	0.99
Learning rate	0.0005
Local buffer size B	512
Update probability P	0.5
Sampling probability α	0.6
Q-learning interval	50
Synchronization interval	1000

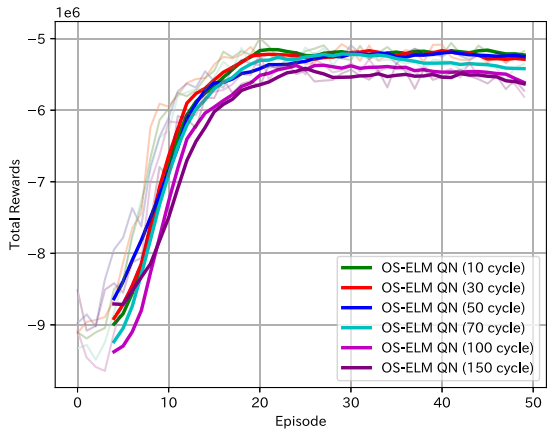

Fig. 7 Total rewards vs. training episodes of OS-ELM QN with different update intervals. Y-axis shows the total rewards of all nodes. Light-colored lines show average rewards. Heavy-colored lines show the 5-moving average.

increase the number of control packets in the network. We thus selected 50 cycles as the update interval of states in this paper.

Regarding the validity range of this paper, first of all we made the following assumptions. The reinforcement learning based route optimization method is running on routing nodes in a network. Their current network states are propagated over the network as well as conventional practical routing protocols, such as OSPF. Such control packets (including those of OSPF) are quite small compared with data traffic. The machine learning time is not integrated in our network simulator. All the evaluations are done ten times and average values of the ten trials are reported.

5.2 Learning Performance

In this section, we compare the performance of a 4-layer DQN and our OS-ELM QN with the three different sampling techniques (a), (b), and (c) in Fig. 2. That is, the following

Table 3 Number of nodes in each layer in DQN and OS-ELM QN models for 4x4 mesh network. Number of shortest paths or actions is 2.

	Input layer	Hidden layers		Output layer
DQN	16	64	64	2
OS-ELM QN	16+2	64		1

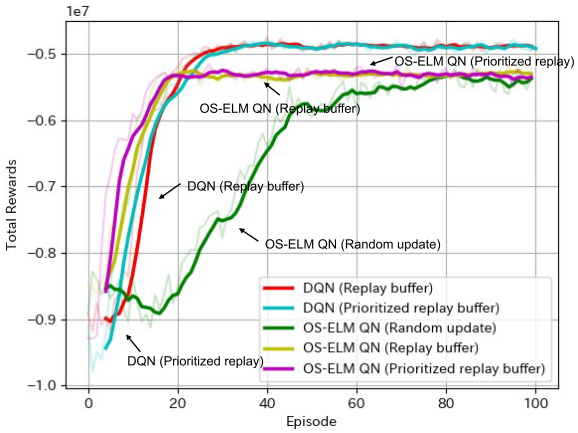

Fig. 8 Total rewards vs. training episodes of different methods.

five methods (listed in Table 1) are compared.

1. 4-layer DQN (Replay buffer)
2. 4-layer DQN (Prioritized replay buffer)
3. OS-ELM QN (Random update)
4. OS-ELM QN (Replay buffer)
5. OS-ELM QN (Prioritized replay buffer)

We set the batch size to 64[†]. The interval of Q-learning is 50 cycles[††]. The synchronization interval between the target Q-Network and the main Q-Network as mentioned in Sect. 2.2 is set to 1000 cycles. The random update probability P is 0.5. The sampling probability parameter α is 0.6. Model parameters of OS-ELM QN are listed in Table 2. Table 3 shows the neural network models of DQN and OS-ELM QN. In this experiment, 8000 packets are injected into a network in an episode. The injection rate is set to 0.5 packets per cycle per node, which means that 8 packets are injected per cycle until 1000 cycles in the 4x4 mesh network. All the evaluations are done ten times and average values of the ten trials are reported in this paper.

Figure 8 shows the total rewards of all the nodes in the network. First, we compare (3) Random update method with the replay buffer methods ((4) and (5)). OS-ELM QN with an experience replay buffer shows a better learning performance than the random update method, which means that

[†]As a preliminary evaluation, OS-ELM QN was evaluated by varying the batch size from 16 to 128. Since there was no significant difference in the results, we selected 64 as a middle of these batch sizes.

[††]OS-ELM QN was evaluated by varying the Q-learning interval from 25 to 100 cycles. A shorter Q-learning interval can achieve a higher reward especially in earlier episodes, while that with 25 cycles slightly decreases the reward in 40 episodes or later; we thus selected 50 cycles as the Q-learning interval.

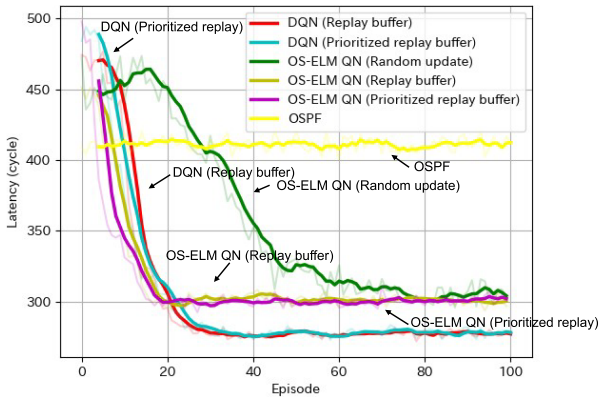

Fig. 9 Packet latency vs. training episodes of different methods. Y-axis shows the total number of cycles to transfer a packet.

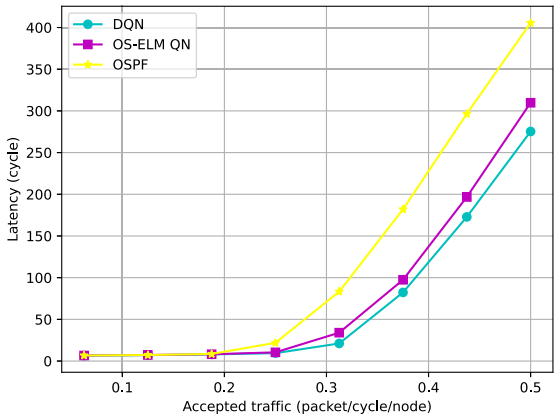

Fig. 10 Packet transfer latency under uniform traffic in 4x4 mesh topology.

the experience replay buffer technique contributes significantly to the performance of OS-ELM QN.

We then focus on performance differences between DQN based methods ((1) and (2)) and OS-ELM QN based methods ((4) and (5)). The OS-ELM QN methods can be trained more efficiently than the DQN methods in the first 20 episodes. This is because the OS-ELM QN methods can optimize parameters in one-shot, while the DQN methods optimize the parameters gradually. However, in subsequent episodes, the DQN methods acquire higher average reward values than the OS-ELM QN methods. This is because the OS-ELM QN methods use a 3-layer neural network and only $\boldsymbol{\beta}$ can be optimized.

Next, we consider the performance between the replay buffer technique ((1) and (4)) and the prioritized replay buffer technique ((2) and (5)). These two techniques show some differences in the early episodes of the training process for both DQN and OS-ELM QN. The total reward values increase quickly for the prioritized replay buffer methods in the first 10 episodes. This result indicates that the prioritized replay technique can optimize the parameters quickly. In the end, these two techniques are converged to the stationary state. In the stationary state, the differences between OS-ELM QN with a replay buffer and that with a prioritized replay buffer are not significant. In our experiments, the number of next hops (i.e., the number of actions) in each node is mostly small. Since the diversity of actions and their outcomes is limited especially in mesh topologies, the benefit of the prioritized replay buffer is also limited.

Finally, we evaluate these methods in terms of the packet transfer latency. Figure 9 shows the relationship between the packet latency and training length in episodes. In addition to the reinforcement learning methods ((1) to (5)), we evaluate the latency of OSPF in the same environment for comparison. Comparing the reinforcement learning methods with OSPF, the latency in the reinforcement learning methods decrease significantly. We set the negative number of the latency as the reward value as proposed in Sect. 4.2.1. Therefore, the total latencies of the reinforcement learning methods decrease as the number of episodes increases. We will discuss the latency more detail in Sect. 5.3.

5.3 Packet Transfer Latency

We evaluate the packet transfer latency under various traffic patterns. We inject uniform, transpose, and bit reverse traffic pattern packets into a network. In the simulations, the packet injection rate is varied from 0 to 0.5 packets per cycle per node in the cases of (2) DQN, (5) OS-ELM QN, and OSPF. We use the DQN and OS-ELM QN models which have been trained with 100 episodes[†] under a uniform traffic pattern[††]. The trained injection rates are 0.5 and 0.125 for the 4x4 mesh and 8x8 mesh networks, respectively. The machine learning time is not integrated in the network simulations. This paper focuses on the machine learning times of OS-ELM QN and DQN. They are separately evaluated in Sect. 5.4.

5.3.1 4x4 Mesh Topology

Figures 10, 11, and 12 show the latencies in uniform, transpose, and bit reverse traffic patterns, respectively. The latency value displayed in these figures is an average packet transfer latency of all the generated packets. Standard deviations of the packet latencies are shown in Tables 4, 5, and 6, respectively.

In Fig. 10, there is little difference in the latency when the injection rate is less than 0.25. However, after that, the two reinforcement learning methods ((2) and (5)) achieve lower latencies than OSPF. These results demonstrate that these reinforcement learning methods choose routing paths

[†]As shown in Figs. 8 and 9, the reinforcement learning methods are fully converged within 100 episodes. We thus select 100 episodes as the pretraining time.

[††]Especially in the uniform traffic, communication occurs in all the source-destination pairs, and traffic workload is balanced over the network. Since an agent is trained for each source-destination pair in the reinforcement learning methods, all the agents can be trained in the uniform traffic pattern. We thus select the reinforcement learning models pretrained by the uniform traffic.

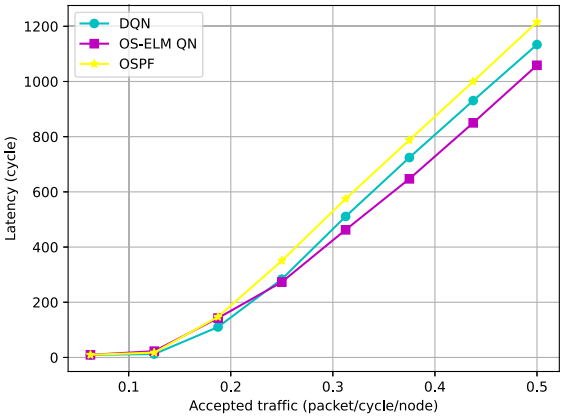

Fig. 11 Packet transfer latency under transpose traffic in 4x4 mesh topology.

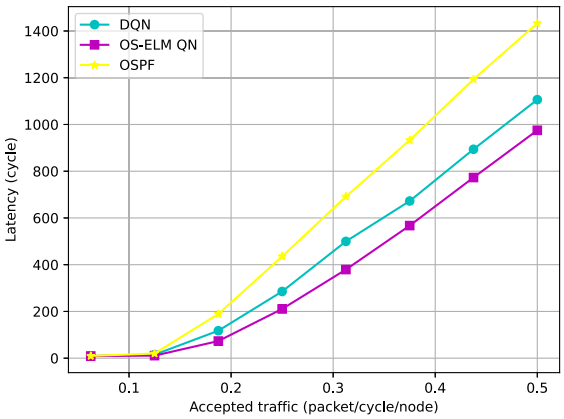

Fig. 12 Packet transfer latency under bit reverse traffic in 4x4 mesh topology.

Table 4 Average and standard deviation of packet latency under uniform traffic in 4x4 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	7.070	9.466	82.30	275.3
	std	4.532	6.018	68.43	211.8
OS-ELM QN	avg	7.265	10.44	97.44	309.7
	std	4.549	6.241	88.87	249.8
OSPF	avg	7.223	21.83	181.9	405.5
	std	4.601	15.73	179.2	339.5

Table 5 Average and standard deviation of packet latency under transpose traffic in 4x4 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	12.18	283.6	724.4	1133
	std	9.811	290.8	552.9	763.3
OS-ELM QN	avg	22.79	273.3	647.2	1059
	std	15.97	138.2	337.0	612.4
OSPF	avg	17.57	350.4	788.1	1215
	std	12.89	296.9	613.3	850.1

intelligently when the packet injection rate is increased.

To further analyze the results, we measure the average number of packets processed in each node during an episode. Uniform traffic is used as well. The packet injection rate is fixed at 0.5. OS-ELM QN and OSPF are compared in terms of the load balancing of packets.

Table 6 Average and standard deviation of packet latency under bit reverse traffic in 4x4 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	14.86	285.3	672.5	1106
	std	8.212	330.7	616.2	805.9
OS-ELM QN	avg	10.90	210.6	567.3	975.1
	std	5.495	169.5	337.4	563.0
OSPF	avg	20.02	435.2	933.2	1433
	std	11.37	294.8	580.8	843.0

1.56	122.76	118.39	1.08
118.46	148.22	189.14	97.20
38.83	215.34	186.21	70.22
1.47	70.40	49.11	1.93

Fig. 13 Average number of packets processed in each node during an episode (OS-ELM QN, 4x4 mesh, uniform traffic).

1.94	79.18	119.29	1.03
127.59	374.83	263.20	12.69
127.20	204.33	129.56	3.42
0.82	1.67	1.37	0.10

Fig. 14 Average number of packets processed in each node during an episode (OSPF, 4x4 mesh, uniform traffic).

Figures 13 and 14 show the results of OS-ELM QN and OSPF, respectively. In these figures, a darker green indicates more packets, which cause a congestion around it. Since a uniform traffic is used in this experiment, the traffic workload should be evenly distributed over all the nodes. In the case of OSPF (Fig. 14), however, the workload is high in central nodes while it is low in corner nodes. In the case of OS-ELM QN (Fig. 13), the traffic workload becomes uniform compared with the OSPF case, demonstrating that OS-ELM QN can balance the workload better than OSPF. In other words, the routing paths are intelligently decided to avoid congestions.

In Fig. 12, the two reinforcement learning methods are better than OSPF. These results also show that since these methods choose routing paths to avoid congestions, they can reduce the latencies. Comparing these two methods, OS-ELM QN reduces more latency compared to DQN. This result shows that the simple model of OS-ELM QN is well optimized to this problem. In Fig. 11, the reinforcement learning methods are slightly better than OSPF, but the differences are small and their benefits are not significant in this pattern.

5.3.2 8x8 Mesh Topology

We evaluate packet transfer latencies in an 8x8 mesh topology. Figures 15, 16, and 17 show latencies in uniform, transpose, and bit reverse traffic patterns, respectively. In

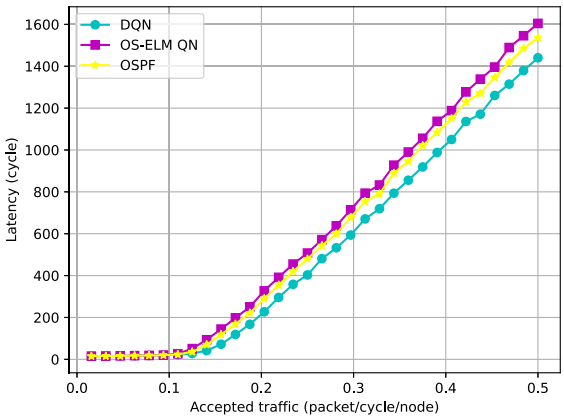

Fig. 15 Packet transfer latency under uniform traffic in 8x8 mesh topology.

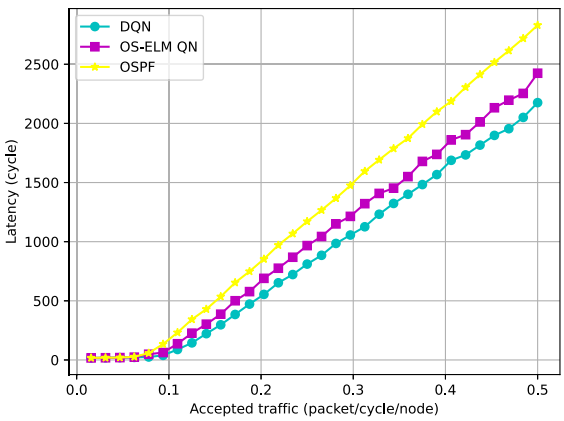

Fig. 16 Packet transfer latency under transpose traffic in 8x8 mesh topology.

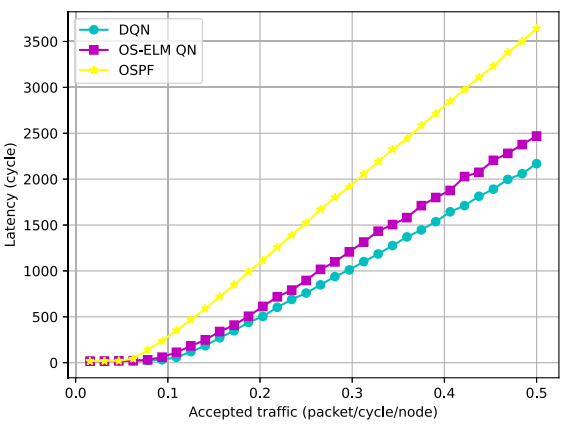

Fig. 17 Packet transfer latency under bit reverse traffic in 8x8 mesh topology.

Fig. 15, latency differences between the three methods are decreased compared to Fig. 10. In Figs. 16 and 17, the reinforcement learning methods achieve lower latencies than OSPF. In Fig. 16, when the injection rate is more than 0.1, latencies of OS-ELM QN are approximately 80% of those of OSPF. In Fig. 17, when the injection rate is more than 0.1, latencies of OS-ELM QN are approximately 70% of those of OSPF.

Table 7 Average and standard deviation of packet latency under uniform traffic in 8x8 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	27.82	415.1	915.6	1447
	std	17.57	318.12	615.1	881.9
OS-ELM QN	avg	55.18	517.5	1054	1616
	std	26.08	426.0	744.8	1038
OSPF	avg	37.69	487.0	1008	1555
	std	34.05	390.1	689.8	972.3

Table 8 Average and standard deviation of packet latency under transpose traffic in 8x8 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	144.3	722.1	1482	2175
	std	135.0	594.5	966.4	1311
OS-ELM QN	avg	226.5	966.9	1679	2424
	std	200.7	771.0	1129	1526
OSPF	avg	343.0	1171	1992	2830
	std	320.6	843.0	1357	1896

Table 9 Average and standard deviation of packet latency under bit reverse traffic in 8x8 mesh topology.

	Injection rate	0.125	0.25	0.35	0.5
DQN	avg	120.0	758.6	1446	2168
	std	90.48	483.9	755.5	1224
OS-ELM QN	avg	182.1	896.1	1711	2469
	std	153.8	622.8	1012	1344
OSPF	avg	468.2	1524	2585	3642
	std	394.2	985.0	1568	2096

Please note that the latency differences between the three methods in the 8x8 mesh are larger than those in the 4x4 mesh except for the uniform traffic. This is because since the network size is increased from 4x4 to 8x8, the number of available routing paths increases and thus benefits of selecting best paths also increase. It is thus expected that a larger network size has more room to reduce transfer latencies. Comparing the two reinforcement learning methods, DQN is superior to OS-ELM QN in the 8x8 network as shown in Fig. 17. In Sect. 5.3.1, we reported that OS-ELM QN reduces more latency compared to DQN in the 4x4 mesh with bit reverse traffic. In the larger network, however, OS-ELM QN is inferior to DQN due to a high generalization performance of DQN. It seems that DQN can take advantage of the high learning ability in the larger network. The network simulation results show that a larger network size has more room to reduce transfer latencies, while the machine learning times increase proportionally to the number of nodes, as we will show in Sect. 5.4.

In addition to the packet transfer latencies shown in Figs. 10-12 and 15-17, their average and standard deviation are also shown in Tables 4-9. The latency value displayed in these figures is an average packet transfer latency of all the generated packets. It is observed that the standard deviation

Table 10 Number of nodes in each layer in DQN and OS-ELM QN models for ARPANET.

	Input layer	Hidden layers		Output layer
DQN	29	64	64	2
OS-ELM QN	29+2	64		1

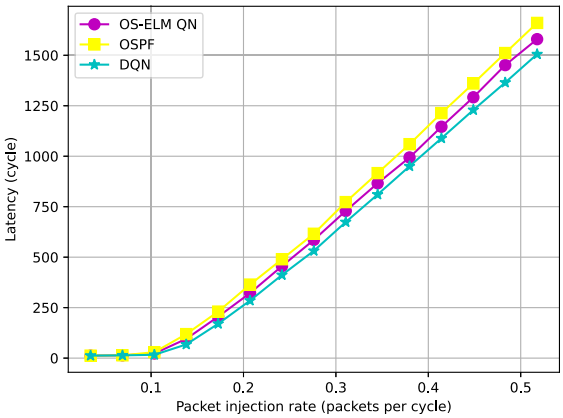

Fig. 18 Packet transfer latency under uniform traffic in ARPANET.

Table 11 Average and standard deviation of packet latency under uniform traffic in ARPANET.

	Injection rate	0.14	0.28	0.38	0.52
DQN	avg	67.79	531.3	950	1505
	std	67.82	450.7	734.0	1096
OS-ELM QN	avg	95.25	585.5	993.9	1579
	std	100.7	531.7	817.1	1200
OSPF	avg	118.9	615.9	1059	1660
	std	127.1	561.5	877.1	1285

increases as the number of packets that take more transfer latency is increased.

5.3.3 ARPANET

In addition, OS-ELM QN, DQN, and OSPF are evaluated on ARPANET (1972) as a realistic topology. Model parameters of OS-ELM QN and DQN are listed in Table 10. Both the OS-ELM QN and DQN use a prioritized replay buffer. These reinforcement learning models were trained for 100 episodes on the ARPANET assuming a uniform traffic. Packets are generated in the first 1000 cycles, and packet generation rate is 0.28 packets per cycle per node (i.e., 8 packets per step). An episode is completed when packet transfers of all the generated packets are completed. These models are evaluated in terms of packet transfer latency.

Figure 18 shows the evaluation results, where the latencies are averaged over ten trials. Standard deviations of the packet latencies are shown in Table 11. In the ARPANET, the number of available paths for each destination is limited compared with the mesh topologies, reducing the benefits of route optimization by the reinforcement learning. The evaluation results show that the performance improvements by the reinforcement learning are limited compared with those in the mesh topologies. Nevertheless, OS-ELM QN is better than OSPF, demonstrating a potential of OS-ELM QN in a practical network.

Table 12 Inference time and learning time (sec) for 4x4 mesh topology on Intel Core i7-10700 CPU.

	Inference time	Learning time
DQN	1326.0	3191.2
OS-ELM QN	1185.1	1367.7
DQN / OS-ELM QN	1.120	2.333

Table 13 Standard deviation of inference time and learning time (sec) for 4x4 mesh topology on Intel Core i7-10700 CPU.

	Inference time	Learning time
DQN	3.310	74.25
OS-ELM QN	25.91	63.76

Table 14 Inference time and learning time (sec) for 8x8 mesh topology on Intel Core i7-10700 CPU.

	Inference time	Learning time
DQN	2674.8	12531
OS-ELM QN	2351.7	5528.5
DQN / OS-ELM QN	1.137	2.267

Table 15 Standard deviation of inference time and learning time (sec) for 8x8 mesh topology on Intel Core i7-10700 CPU.

	Inference time	Learning time
DQN	10.45	102.1
OS-ELM QN	32.01	223.7

5.4 Learning Time

In this section, we evaluate the learning time of OS-ELM QN and DQN. The evaluation condition is the same as that in Sect. 5.2. Table 12 shows the inference and learning times of the DQN and OS-ELM QN methods in 100 episodes. Their standard deviations are shown in Table 13.

The values in the table are in seconds. The running time of the network simulator and experience sampling are not included. The learning time results show that the OS-ELM QN method is about 2.33 times faster than the DQN method. This is because OS-ELM QN does not use the backpropagation algorithm but one-shot optimization. As a result, the OS-ELM QN model can be trained faster.

The inference and learning times in the 8x8 network are shown in Table 14. Their standard deviations are shown in Table 15. Eight packets are injected in each cycle, which means 0.125 packets per cycle per node. These results show that the execution times increase in proportional to the number of nodes. The speedup ratios by the OS-ELM QN against the DQN are almost constant when the network size is changed from 4x4 to 8x8.

In this paper, we assume the proposed reinforcement learning based routing optimization method is implemented on intelligent network equipment. To make this assumption more realistic, we implement the proposed OS-ELM QN

Table 16 Specification of Bluefield-2 DPU.

CPU	Eight Armv8 A72 cores (64-bit) @ 2.5GHz
NIC	Two 25GbE I/Fs (2xSFP56)
PCIE	Gen 4.0 x8
DDR4 Memory	16GB
eMMC Memory	64GB

Table 17 Average of inference time and learning time (sec) for 4x4 mesh topology on NVIDIA Bluefield-2 DPU.

	Inference time	Learning time
DQN	10212	41017
OS-ELM QN	6661	4784
DQN / OS-ELM QN	1.533	8.574

Table 18 Standard deviation of inference time and learning time (sec) for 4x4 mesh topology on NVIDIA Bluefield-2 DPU.

	Inference time	Learning time
DQN	31.50	616.9
OS-ELM QN	48.12	78.08

algorithm on NVIDIA Bluefield-2 DPU as a commercial network processing platform. Table 16 shows the specification of the DPU used in this experiment, respectively.

We evaluate OS-ELM QN and DQN in terms of training and inference times on the DPU. A 4x4 mesh topology is used. Model parameters are the same as those in Table 3. Table 17 shows the evaluation results, where the execution times are averaged over ten trials. Table 18 shows their standard deviations. The results show that the inference and training of OS-ELM QN are faster than DQN by 1.533 times and 8.574 times, respectively. These results demonstrate benefits of the computation cost reduction by the proposed OS-ELM QN especially in such network processing platform. Please note that both the DQN and OS-ELM QN methods invoke the inference and training functions implemented with Tensorflow. Since the inference function is called more frequently compared with the training function during the 100 episodes, invocation costs of Tensorflow become dominant especially in the inference time of OS-ELM QN in DPU.

6. Discussion

This section discusses the evaluation results of OS-ELM QN and its related techniques.

First, we compare OS-ELM QN with DQN in packet routing. In terms of the learning performance, while OS-ELM QN performed better than DQN in the early stages of learning, the final reward of DQN was better than OS-ELM QN in Fig. 8. These results imply that OS-ELM QN can adapt to traffic patterns quickly, while DQN still provides a stable and high-performance routing. In terms of the learning time, OS-ELM QN could be trained 2.33 times faster than DQN. OSPF showed a lower transfer efficiency than the reinforcement learning methods, though OSPF does not require a training time of neural networks. These results demonstrated that there is a tradeoff between performance and learning cost.

The three sampling techniques were compared in this paper. In this environment, although the random update technique did not work well, the experience replay and prioritized experience replay techniques improved the learning efficiency. Especially, the prioritized experience replay technique achieved a higher efficiency than the experience replay technique in the first 10 episodes of Fig. 8.

Regarding the reinforcement learning settings (e.g., state, action, reward) introduced in Sect. 4.2.1, we consider that these settings worked sufficiently based on the results from Fig. 9 and Sect. 5.3. Further tuning on the reinforcement learning settings is our future work.

In this paper, each node has $(N-1)$ neural networks, so the number of neural networks in a network is quadratically increased when the network size is increased. To alleviate this problem, we are considering to combine the state (i.e., buffer occupancies) and one-hot encoded destination node as an input state. This modification may affect the accuracy while it can compress the number of neural networks in each node.

7. Conclusions

In this paper, we proposed OS-ELM QN as an OS-ELM based reinforcement learning method for intelligent packet routing. We evaluated the performance of OS-ELM QN by using a network simulator to measure the packet transfer latency. Compared to an existing work [8], we changed the sampling technique and input structure of neural networks. Experimental results showed that introducing the experience replay buffer improves the learning performance. OS-ELM QN achieves a 2.33 times speedup than DQN in terms of the learning speed. Regarding the packet transfer latency, OS-ELM QN is comparable or slightly inferior to DQN while they are better than OSPF in most cases since they can distribute traffic congestions.

References

[1] "Cisco Annual Internet Report." https://www.cisco.com/c/ja_jp/solutions/executive-perspectives/annual-internet-report/index.html (Accessed on 12/15/2022).
[2] J. Moy, "OSPF Version 2," RFC 2178, July 1997.
[3] C.L. Hedrick, "Routing Information Protocol," RFC 1058, June 1988.
[4] N. Kato, Z.M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," IEEE Wireless Commun., vol.24, no.3, pp.146–153, 2017.
[5] Y. Xiao, J. Liu, J. Wu, and N. Ansari, "Leveraging Deep Reinforcement Learning for Traffic Engineering: A Survey," IEEE Communications Surveys & Tutorials, vol.23, no.4, pp.2064–2097, 2021.
[6] R. Ding, Y. Xu, F. Gao, X. Shen, and W. Wu, "Deep reinforcement learning for router selection in network with heavy traffic," IEEE Access, vol.7, pp.37109–37120, 2019.
[7] N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan, "A Fast and Accurate Online Sequential Learning Algorithm for

Feedforward Networks," IEEE Trans. Neural Netw., vol.17, no.6, pp.1411–1423, Nov. 2006.
[8] H. Watanabe, M. Tsukada, and H. Matsutani, "An FPGA-Based On-Device Reinforcement Learning Approach using Online Sequential Learning," Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS'21) Workshops, pp.96–103, May 2021.
[9] K. Nemoto and H. Matsutani, "A Packet Routing using Lightweight Reinforcement Learning Based on Online Sequential Learning," Proc. of the 10th International Symposium on Computing and Networking (CANDAR'22) Workshops, Nov. 2022.
[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," NeurIPS Deep Learning Workshop 2013, 2013.
[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," Proc. International Conference on Learning Representations (ICLR'16), May 2016.
[12] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to Route," Proc. 16th ACM Workshop on Hot Topics in Networks, pp.185–191, 2017.
[13] "Gym." https://gym.openai.com/ (Accessed on 12/15/2022).
[14] "NetworkX." https://networkx.org/ (Accessed on 12/15/2022).

Kenji Nemoto received the BE degree from Keio University in 2022. He is currently a master course student in Keio University.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is currently a professor in the Department of Information and Computer Science, Keio University. His research interests include the areas of computer architecture and interconnection networks.