Efficient Compressed Ratio Estimation Using Online Sequential Learning for Edge Computing

Hiroki Oikawa, Hangli Ge, Noboru Koshizuka
Graduate School of Interdisciplinary Information Studies,
The University of Tokyo,
Tokyo, Japan
{hiroki.oikawa, hangli.ge, noboru}@koshizuka-lab.org

Abstract—Owing to the widespread adoption of the Internet of Things, a vast amount of sensor information is being acquired in real time. Accordingly, the communication cost of data from edge devices is increasing. Compressed sensing (CS), a data compression method that can be used on edge devices, has been attracting attention as a method to reduce communication costs. In CS, estimating the appropriate compression ratio is important. There is a method to adaptively estimate the compression ratio for the acquired data using reinforcement learning (RL). However, the computational costs associated with existing RL methods that can be utilized on edges are often high. In this study, we developed an efficient RL method for edge devices, referred to as the actor–critic online sequential extreme learning machine (AC-OSELM), and a system to compress data by estimating an appropriate compression ratio on the edge using AC-OSELM. The performance of the proposed method in estimating the compression ratio is evaluated by comparing it with other RL methods for edge devices. The experimental results indicate that AC-OSELM demonstrated the same or better compression performance and faster compression ratio estimation than the existing methods.

Index Terms—edge computing, data compression, IoT

I. INTRODUCTION

The widespread use of Internet of Things (IoT) devices has led to the acquisition of diverse sensor information in real time. Utilizing such sensor information, technologies such as artificial intelligence have made significant progress. However, with the increase in the number of IoT devices, the volume of data communication is also increasing, resulting in immense power and communication bandwidth costs [1].

Data compression technology is used to reduce the cost of data transmission by reducing the amount of data without compromising its features. In particular, compressed sensing (CS) has garnered attention as a method that can be used on edge devices with low computational resources [2]. In CS, it is important to determine an appropriate compression ratio for the acquired data considering the trade-off between the reduction of data transmission and reconstruction errors.

A method exists for estimating the compression ratio in CS using reinforcement learning (RL). Sekine et al. proposed a method for estimating the optimal compression ratio using deep RL (DRL) for edge devices [3]. DRL is an RL method that uses deep neural networks (DNN) and has made significant progress in recent years [4]. However, edge devices with fewer computational resources, such as sensor nodes, have difficulty incorporating DRL, which is computationally expensive. Therefore, a more efficient RL method is required to estimate the compression ratio on edge devices such as sensor nodes.

There exist efficient RL methods for edge devices that employ the online sequential extreme learning machine (OS-ELM) [5]. The OS-ELM is a single-layer neural network capable of fast sequential learning. Watanabe et al. proposed the OS-ELM-Q-Network (OS-QNet) as an RL method with low computational cost, based on OS-ELM [6]. They demonstrated that OS-QNet could be efficiently executed on field-programmable gate arrays (FPGAs). However, in OS-QNet, the cost of inferring the optimal action becomes large when there are many action patterns, such as the compression ratio. Therefore, the estimation of compression ratios using OS-QNet at the edge becomes a computational bottleneck in data compression using CS.

This study proposes an efficient RL method referred to as the actor–critic OS-ELM (AC-OSELM). An actor–critic is an RL method that sequentially trains an actor to learn a policy and a critic to estimate the value of a state [7]. In the proposed AC-OSELM, the actor and critic, which are both composed of single-layer neural nets, are updated with a deterministic policy gradient (DPG) [8] and OS-ELM, respectively, to achieve RL with a low computational cost. AC-OSELM can efficiently infer optimal actions even when the number of action patterns is large, such as the compression ratio. Therefore, we propose a system that uses AC-OSELM to estimate the compression ratio in CS. This system allows efficient data compression on edge devices with low computational resources.

The contributions of this study are summarized as follows:

- We developed AC-OSELM as an RL method for edge devices. Unlike the existing OS-QNet, AC-OSELM can be used efficiently when there are several action patterns.
- We developed AC-OSELM-based compression ratio estimation system for CS. This system has a low computational cost and is suitable for execution on edge devices.
- We conducted experiments to evaluate the performance of the compression ratio estimation in CS using AC-OSELM and compared it with OS-QNet.

The experimental results indicated that AC-OSELM achieved the same or better compression performance than OS-QNet for every dataset. In addition, AC-OSELM significantly
reduced the computational cost of inference compared with OS-QNet, despite a higher computational cost of learning. The compression ratio estimation model for edge devices requires higher real-time performance for inference than for learning. Therefore, the proposed compression ratio estimation method based on AC-OSELM can be considered suitable for edge-device implementation.

II. RELATED WORK

Various studies have been conducted on data compression on edge devices. Azar et al. proposed an efficient compression scheme using Squeeze, which is a fast error-bounded lossy compression scheme [9], for IoT devices [10]. CS is a method that can be applied to sparse data [2]. This compression method achieves accurate data recovery and low compression ratios for sparse data. Owing to its low computational cost, CS has been used in applications wherein real-time performance is required, such as image communication systems for IoT monitoring applications at sensor nodes [11]. Li et al. also proposed a power- and communication-efficient framework for wireless sensor networks and IoT devices using CS [12]. CS requires an appropriate compression ratio for the data to perform both data reduction and reconstruction properly. A system that uses a DNN on edge devices to estimate the compression ratio has been proposed previously [3]. Using RL, this system can learn and estimate an appropriate compression ratio for the acquired data. However, owing to the high computational costs associated with DNNs, the model cannot be used on edge devices with low computational resources. Therefore, efficient RL models are required for use at the network edge.

As mentioned, OS-ELM is a neural network with a low computational cost that follows changes in data trends and has a fast sequential learning capability for data acquired in a sequence [5]. It can achieve high-speed processing on FPGAs, making it suitable for use on edge devices [13]. OS-QNet is an RL method that uses the OS-ELM [6]. This method is known to be computationally efficient and significantly faster than the existing Deep Q-Network (DQN) [4] when implemented on FPGAs. However, the computational cost of selecting actions in OS-QNet increases in proportion to the number of actions, and it cannot handle continuous action spaces, such as the compression ratio. Therefore, this study proposes AC-OSELM as an OS-ELM-based RL method that can be used even when the pattern of actions is not finite. In addition, we propose AC-OSELM-based system for estimating the compression ratio in CS that can be applied to edge devices with small computational resources.

Data extraction is a method for reducing the data volume at the edge. This method reduces the data sent to the cloud server by selecting data from edge devices. Papageorgiou et al. proposed a system for efficiently extracting acquired time series data by automatically switching between multiple data handlers [14]. In addition, a method exists for extracting the representative data from a large data stream by using submodular optimization [15]. Such methods that directly reduce the number of data to be transmitted can be used in conjunction with data compression methods that can reduce the cost of transmission without reducing the quantity of data.

III. PRELIMINARIES

A. Compressed Sensing

CS is a data compression method proposed by Donoho [2]. This method compresses the acquired n-dimensional data $x \in \mathbb{R}^{n \times 1}$ into m-dimensional data $y \in \mathbb{R}^{m \times 1}$ using a random matrix $\Phi \in \mathbb{R}^{m \times n}$, as shown in the following equation:

$$y = \Phi x$$

(1)

Because $m < n$, determining the value of $x$ from the values of $y$ and $\Phi$ is usually impossible. We assume that $x$ is sparse in a certain feature space. Then, using the transformation matrix $\Psi \in \mathbb{R}^{n \times n}$ and sparse vector $x_s \in \mathbb{R}^{n \times 1}$, $x$ can be expressed as follows:

$$x = \Psi x_s$$

(2)

The value of $x_s$ can be obtained by solving the L1-norm minimization problem for $x_s$, as follows:

$$\text{minimize} \ |x_s|_1$$
$$\text{subject to} \ y = \Phi \Psi x_s$$

(3)

(4)

Using $x_s$, we can recover the value $x$ of the acquired data as $x = \Psi x_s$. Thus, CS can achieve fast compression for sparse data in a certain feature space. This is a suitable method for implementation on edge devices wherein fast processing is required to maintain real-time performance.

B. Reinforcement Learning

In RL, an agent in an environment observes its current state and learns an action strategy (policy) that maximizes the reward obtained from the environment for the observed state.

C. OS-QNet

Q-learning is an RL method estimating the effectiveness of an action $a_t$ in a certain state $s_t$ using the Q-function $Q(s_t, a_t)$. In Q-learning, $Q(s_t, a_t)$ is sequentially updated, and for state $s_t$, the action $a_t$ that maximizes $Q(s_t, a_t)$ is selected. DQN is a method for constructing this Q-function using DNN [4]. Although DQN demonstrates superior generalization performance, it incurs a high computational cost and is not suitable for execution on edge devices. Therefore, OS-QNet, a Q-learning method using the OS-ELM, has been proposed [6].
1) OS-ELM: Algorithm of ELM: The ELM is a single-layered neural network model that can be trained analytically [16]. It has the following parameters: $\alpha$, the weight from the input layer to the hidden layer; $b$, the bias of the hidden layer; and $\beta$, the weight from the hidden layer to the output layer. Only $\beta$ is updated during the learning process, while the other values are fixed. In the training process, the value of $\beta$ that minimizes the following value $L$ for input data $X$ and target data $Y$ is calculated:

$$L = \|H\beta - Y\|$$

(5)

$$H = g(X\alpha + b)$$

(6)

where $g(\cdot)$ denotes the activation function of the hidden layer. $\hat{\beta}$, which minimizes $L$, can be obtained analytically using the following equation:

$$\hat{\beta} = H^\dagger Y$$

(7)

where $H^\dagger$ is the Moore–Penrose pseudo inverse of $H$. A method to train the Q-function and the critic function (to be described later) with ELM has been proposed and is known to be efficient for learning [17], [18].

Algorithm of OS-ELM: The OS-ELM is a model that extends ELM to sequential learning [5]. Let the sequentially acquired data be denoted as $(X_i, Y_i) (i = 0, 1, \ldots)$, $\hat{\beta}_i$ where the weights from the hidden layer to the output layer in the OS-ELM are updated, as shown in the following equation:

$$P_0 = (H_0^T H_0)^{-1}, \quad \hat{\beta}_0 = P_0 H_0^T Y_0$$

(8)

$$P_{i+1} = P_i - P_i H_{i+1}^T (\lambda I + H_{i+1} P_i H_{i+1}^T)^{-1} H_{i+1} P_i$$

(9)

$$\hat{\beta}_{i+1} = \hat{\beta}_i + P_{i+1} H_{i+1}^T (Y_i - H_{i+1} \hat{\beta}_i)$$

(10)

where $\lambda$ denotes the forgetting rate, which is a constant to be set in advance.

2) Algorithm of OS-QNet: Figure 1 depicts an overview of OS-QNet. In OS-QNet, the Q-function is constructed using OS-ELM. This algorithm is shown in Algorithm 1 where $\alpha$, $b$, $\beta$, and $N$ correspond to the weight from the input layer to the hidden layer, bias in the hidden layer, weight from the hidden layer to the output layer, and the update period, respectively. $\epsilon(t)$ is the probability of choosing a random action and decays as $t$ increases. The update of $\beta$ in Algorithm 1 is given by the following equation. When $t = N$, it is updated according to Eq.(11); otherwise, it is updated according to Eq.(12,13).

$$P = (H^T H)^{-1}, \quad \beta = P H^T R$$

(11)

$$P = P - P H^T (\lambda I + H P H^T)^{-1} H P$$

(12)

$$\beta = \beta + P H^T (R - H \beta)$$

(13)

where $H$ and $R$ are computed using Eq.(14) using $(a_t, s_t, r_t)$ in $D$.

$$H = \begin{pmatrix} h|_{s=s_1, a=a_1} \\ \vdots \\ h|_{s=s_N, a=a_N} \end{pmatrix}, \quad R = \begin{pmatrix} r_1 \\ \vdots \\ r_N \end{pmatrix}$$

(14)

where $h|_{s=s_t, a=a}$ is the output of the hidden layer of OS-ELM for $(s_t, r_t)$ and corresponds to $g((s_t, r_t)\alpha + b)$ using the activation function $g(\cdot)$.

At a low computational cost, OS-QNet can learn based on this procedure and be implemented on edge devices. However, when estimating the optimal action for a state, all action patterns must be evaluated with a Q-function. Therefore, OS-QNet cannot handle continuous action spaces, such as compression ratio, which is a continuous value. In addition, even if there are a finite number of action patterns, when there are several, the computational cost when inferring the action becomes large.

IV. ACTOR–CRITIC ONLINE SEQUENTIAL EXTREME LEARNING MACHINE (AC-OSELM)

In this study, we developed AC-OSELM, a novel RL method for edge devices, and a system to compress data by estimating an appropriate compression ratio on the edge using AC-OSELM.

Algorithm 1 Algorithm of OS-QNet

Initialize parameters $\theta = \{\alpha, \beta, b\}$ using random values $\mathbb{R} \in [0, 1]$, buffer $D = \{\}$, global step $t = 0$

for episode $\in \{1, 2, \ldots\}$ do

Observe($s_t$)

while $s_t \neq \text{end}$ do

$t \leftarrow t + 1, p \sim U(0, 1)$

if $p < \epsilon$ then

$a_t \leftarrow \text{random action value}$

else

$a_t = \arg\max_{a \in A} Q(s_t, a)$

end if

end while

Observe($s_{t+1}, r_t$) from environment

$r_t \leftarrow r_t + \gamma \max_{a \in A} Q(s_{t+1}, a)$

Store($a_t, s_t, s_{t+1}, r_t$) in $D$

if $t \% N = 0$ then

Update $\beta$ using $D$ according to Eq.(11, 12, 13)

$D = \{\}$

end if

end for
Algorithm of the proposed AC-OSELM

The actor–critic is a method that simultaneously learns the actor \( \mu(s_t) \) that determines the action, and the critic \( Q(s_t, a_t) \) that evaluates the effectiveness of the policy [7]. In general, actor–critic methods are known to make the model robust to noise and stabilize learning.

In the proposed AC-OSELM, the actor and critic are both consist of a single-layer neural network, as shown in Figure 2. For both the actor and critic, the weights from the input layer to the hidden layer are fixed, and only the weights from the hidden layer to the output layer are updated. The critic and actor are updated using OS-ELM and DPG, respectively.

Let the dimensionality of the state, the dimensionality of the action, the size of the hidden layer, the activation function in the hidden layer, and the activation function in the output layer be \( D, k, m_a, g_{ah}(\cdot) \), and \( g_{ao}(\cdot) \), respectively. Using \( s \in \mathbb{R}^{1 \times D}, a \in \mathbb{R}^{1 \times k} \), function of actor \( \mu_{a}(s) \) is as follows:

\[
\begin{align*}
\h_a &= g_{ah}(s \alpha_a + b_a) \in \mathbb{R}^{1 \times m_a} \\
\mu_{a}(s) &= g_{ao}(h_a b_a) \in \mathbb{R}^{1 \times k}
\end{align*}
\]

where \( \alpha_a, b_a \), and \( \beta_a \) correspond to the weights from the input to the hidden layer of the actor, bias in the hidden layer, and weights from the hidden layer to the output layer, respectively.

Let the size of the hidden layer and activation function in the hidden layer be \( m_c \) and \( g_{ch}(\cdot) \), respectively. Using \( a \in \mathbb{R}^{1 \times k}, \alpha_c \in \mathbb{R}^{(D+k) \times m_c}, b_c \in \mathbb{R}^{1 \times m_c} \), and \( \beta_c \in \mathbb{R}^{m_c \times 1} \), function of critic \( Q_{b_c}(s, a) \) is as follows:

\[
\begin{align*}
x &= (s, a) \in \mathbb{R}^{1 \times (D+k)} \\
h_c &= g_{ch}(x \alpha_c + b_c) \in \mathbb{R}^{1 \times m_c} \\
Q_{b_c}(s, a) &= h_c \beta_c \in \mathbb{R}^{1 \times 1}
\end{align*}
\]

where \( \alpha_c, b_c \), and \( \beta_c \) correspond to the weights from the input layer to the hidden layer of the critic, the bias in the hidden layer, and weights from the hidden layer to the output layer, respectively.

The algorithm for AC-OSELM implementation is described in Algorithm 2. Here, \( N \) denotes the update period. \( \epsilon \) is the noise added to the value of the action output from the actor and decays as \( t \) increases.

\( \beta_a \) and \( \beta_{\delta a} \) in Algorithm 2 are updated as follows. First, \( \beta_c \) is updated according to the OS-ELM using Eq. (11–13). This is the same as the update of the Q-function of OS-QNet.

\[\beta_a \text{ is updated according to DPG. In DPG, the parameter } \theta_c \text{ of } Q \text{ is fixed and the parameter } \theta_a \text{ of } \mu \text{ changes with the gradient of increasing } Q(s, \mu(s)) [8]. \text{ This gradient is given by the following equation:}
\]

\[
\frac{\partial Q(s, a|\theta_c)}{\partial \theta_c} = \frac{\partial a}{\partial \theta_c} \frac{\partial Q(s, a|\theta_c)}{\partial a}
\]

\[
\approx \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mu(s|\theta_a)}{\partial \theta_a} \bigg|_{s=s_i} \frac{\partial Q(s, a|\theta_c)}{\partial a} \bigg|_{s=s_i, a=\mu(s|\theta_a)}
\]

In AC-OSELM, the weights from the input layer to the hidden layer \( \alpha_a \) and the bias in the hidden layer \( b_a \) are fixed, and the weight from the hidden layer to the output layer \( \beta_a \) is updated according to DPG. The formula for updating \( \beta_a \) is as follows. Here, \( \eta \) is the update rate.

\[
\frac{\partial y}{\partial \beta_a} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial y}{\partial \beta_a} \bigg|_{s=s_i} = d \beta_a
\]

\[
\beta_a = \beta_a + \eta d \beta_a
\]

\[
\frac{\partial y}{\partial \beta_a} \text{ is calculated by the following equation, where } \otimes \text{ corresponds to the Hadamard product.}
\]

\[
\frac{\partial y}{\partial \beta_a} = h_a^T g_{ao}'(h_a b_a) \otimes (g_{ch}'(\alpha_c + b_c)(\beta_c \otimes \alpha_c^T))
\]

\[
\alpha_c = \begin{pmatrix}
\alpha_c(D+1)1 & \cdots & \alpha_c(D+1)m_c \\
\vdots & \ddots & \vdots \\
\alpha_c(D+k)1 & \cdots & \alpha_c(D+k)m_c
\end{pmatrix}
\]

By updating only \( \beta_a \) in this manner, the computational cost associated with the actor update is small. In particular, as \( k \) is 1 in the compression ratio estimation, \( g_{ao}'(h_a b_a) \) and \( g_{ch}'(\alpha_c + b_c)(\beta_c \otimes \alpha_c^T) \) becomes scalars, and \( d \beta_a \) can be easily computed.

Thus, AC-OSELM can perform RL while incurring a low computational cost, which is suitable for use on edge devices.

Algorithm 2 Algorithm of AC-OSELM

Initialize parameters \( \theta_a = \{ \alpha_a, b_a, \beta_a \}, \theta_c = \{ \alpha_c, b_c, \beta_c \} \) using random values \( R \in [0, 1] \), buffer \( D = \{ \}, \) global step \( t = 0 \) for episode \( \in \{ 1, 2, \ldots \} \) do

Observe \( s_1 \)

while \( s_t \neq \text{end} \) do

\[
\begin{align*}
t &\leftarrow t + 1 \\
\alpha_c &\leftarrow \mu_{ao}(s_t) + \epsilon (\epsilon \sim P(\epsilon(t))) \\
\beta_a &\leftarrow \mu_{ch}(s_t, \mu_{ao}(s_t))
\end{align*}
\]

Observe \( a_t, r_t \) from environment

\[
\begin{align*}
r_t &\leftarrow r_t + \gamma Q_{ao}(s_{t+1}, \mu_{ao}(s_{t+1})) \\
\text{Store } &\{a_t, s_t, s_{t+1}, r_t, D \}
\end{align*}
\]

if \( t \% N = 0 \) then

Update \( \beta_c \) using \( D \) according to Eq. (11–13)

Update \( \beta_a \) using \( D \) according to Eq. (22–25)

\( D = \{ \} \)

end if

end while

end for
In addition, unlike OS-QNet, AC-OSELM can handle the continuous action spaces and achieve fast action selection even with a large number of action patterns.

B. Application

This paper proposes using AC-OSELM to estimate the compression ratio in CS. This allows for efficient data compression at the edges. As stated, in CS, determining the appropriate compression ratio is necessary. In particular, when processing on the edge, efficient communication can be achieved by adaptively changing the compression ratio according to the data being acquired. However, a trade-off exists between the compression ratio and the data reconstruction error. The smaller the compression ratio, the smaller the data communication cost; however, the data reconstruction error would increase. Considering this trade-off, when estimating the compression ratio \( c \) of CS by RL, the following function \( E(c) \) was proposed as the data transmission efficiency [3].

\[
E(c) = k_1(-k_2c^k_3 + k_4 - k_5c^k_6)
\]

\[
e(c) = \max(0, \text{RMSE}(x, \text{dec}(x_c)) - e_{\text{thr}})
\]

where \( x, x_c, \) and \( \text{dec}(x_c) \) corresponds to the original data, the data compressed \( x \) with compression ratio \( c \), and the data recovered from \( x_c \), respectively. \( k_i (i = 1, 2, ..., 6) \) and \( e_{\text{thr}} \) are constant values that contribute to whether the reduction in the amount of data or the reduction of the reconstruction error is more important, and it is set according to the data and the application. Figure 3 depicts the original image in the MNIST dataset [19] and images recovered from the compressed data at each compression ratio when \([k_1, k_2, k_3, k_4, k_5, k_6, e_{\text{thr}}] = [1, 1, 3, 1, 1, 5, 1, 0.01] \). The value of \( k_i \) was determined such that the range of \( E(c) \) was set to \([0, 1]\) by referring to the value proposed in a prior work [3]. The score is maximum when the compression ratio is 0.50, which is a smaller value among the compression ratios that can properly restore the original data. These results show that \( E(c) \) functions properly as an evaluation index for the compression performance.

Figure 4 shows the overview of the proposed AC-OSELM-based compressed ratio estimation. By training AC-OSELM with the acquired data as the input state, the compression ratio as the action, and \( E(c) \) as the reward, the model to estimate the appropriate compression ratio according to the acquired data can be realized.

![Image](image1)

(a) the original (b) \( c = 0.25 \), (c) \( c = 0.50 \), (d) \( c = 0.75 \), image

\( E(c) = 0.493 \) \( E(c) = 0.875 \) \( E(c) = 0.578 \)

Fig. 3: The original image and images recovered from the compressed data at each compression rate

V. Evaluation

A. Experimental Setup

Procedure: We evaluated the performance of AC-OSELM in estimating the compression ratio in CS compared with that of OS-QNet. Each model acquires and compresses data sequentially in the experiment and is trained after every 10 data acquisitions. The acquisition of these 10 pieces of data is considered one step, and the compression accuracy at each step of each model is evaluated for 2,000 steps. Each RL method was trained using \( E(c) \) \([k_1, k_2, k_3, k_4, k_5, k_6, e_{\text{thr}}] = [1, 1, 3, 1, 1, 5, 1, 0.01] \) as the reward. We also compared the computational cost of each method to measure the computational time required to estimate the compression ratio and learn the model. In AC-OSELM, the range of the compression ratio \( c \) is set to \([0, 1]\), and in OS-QNet, it is set to \([0.1, 0.2, ..., 1.0]\). The number of nodes in the middle layer and the forgetting rate of all models used in the experiment were set to 400 and 0.999, respectively. In data compression, the random matrix \( \Phi \) was generated using \( \mathcal{N}(0, 1) \) and was fixed.

Dataset: For the datasets, we used 100 data elements extracted from MNIST [19] and Kuzushiji-MNIST (KMNIST) [20]. These data are clearly sparse and meet the requirements of CS. Figure 5 shows the transmission efficiency \( E(c) \) of all data at each compressed ratio in each dataset. Data taken at random from the dataset were used as input for each step, and the mean \( E(c) \) of all data in the dataset was used to evaluate the compression accuracy at each step (To evaluate the performance of RL, the scores for all input states were used to evaluate the model).

Environment: To evaluate the utility of the model in sensor nodes, in this experiment, we used Raspberry Pi3 [21] (Arm Cortex-A53 CPU with a 1.2 GHz clock frequency and 1 GB 450 MHz DDR2 main memory). In addition, Python 3.9.2 was used for implementation, and linear programming using PuLP [22] was adopted for data reconstruction. Because the accuracy of the model was not dependent on the device, the compression accuracy was evaluated on a MacBook Air (Apple M1, 8Core, 16 GB) [23] to improve the efficiency of the experimental time.

B. Evaluation Result

Compression Accuracy: Figure 6 shows the compression accuracy (the mean \( E(c) \) of all data in the dataset) at each step when the compression ratio was estimated by each method for each dataset. In both datasets, the accuracy of each method increased as the steps progressed, indicating that each method...
learned appropriately. OS-QNet converged rapidly because its weights were learned analytically. As the steps progressed, the accuracy of AC-OSELM exceeded that of OS-QNet on each dataset. This is because AC-OSELM can estimate the compression ratio more finely than OS-QNet. (The finer the range of compression ratios of OS-QNet, the compression accuracy may become greater, but the computational cost required to estimate the compression ratio increases. The compression ratio estimation with high computational cost is not suitable for compression at the edge that requires real-time performance.) Table I lists the maximum compression accuracy from the last to the 10th step for each method.

### Estimation Example
Figure 7 presents the data transmission efficiency $E(c)$ and the estimated compressed ratio using a trained model of each method for the one piece of data from MNIST (as shown in Figure 3). Although the Q-value of OS-QNet and the critic value of AC-OSELM are close (the interval of OS-QNet is rough), AC-OSELM achieves a higher $E(c)$ than OS-QNet. This is because AC-OSELM could have a wider selection range of compression ratios and estimate a compression ratio closer to the optimal value.

### Computational Cost
Table II lists the computational costs associated with the inference and learning for each method. Here, $|A|$, $D$, $k$, and $n$ indicate the number of action patterns, the number of dimensions of the state, the number of dimensions of the action, and the number of

---

### Table I: Maximum compression accuracy and average compression accuracy from the last to the 10th step

| Dataset | AC-OSELM | OS-QNet | Fixed-ratio |
|---------|----------|---------|-------------|
| MNIST   | 0.930 / 0.923 | 0.921 / 0.921 | 0.908 (c=0.432) |
| KMNIST  | 0.844 / 0.841 | 0.838 / 0.838 | 0.802 (c=0.541) |

### Table II: Computational cost of each method

| Method | Time Complexity | Calculation Time (ms) |
|--------|-----------------|-----------------------|
|        |                 | MNIST | KMNIST |
| Compression | $O(L)$ | 1.14 | 1.38 |
| Recovering  | $O(D^3)$ | 7.35$\times10^4$ | 9.79$\times10^4$ |
| Inference:  | $O(|A|(L^2 + k^2))$ | 2.71$\times10^4$ | 2.73$\times10^4$ |
| OS-QNet     | $O(D^2 + k^2)$ | 2.72 | 2.74 |
| AC-OSELM    |                  |       |       |
| Learning:   | $O(n^3 + nDk + nD^2)$ | 6.74$\times10^4$ | 6.76$\times10^4$ |
| OS-QNet     |                  | 1.02$\times10^2$ | 1.03$\times10^2$ |
| AC-OSELM    |                  |       |       |
training data pieces, respectively. Although the computation time of AC-OSELM during learning increased by up to 52.4% \((= 1.03 \times 10^3 - 1)\) compared to that of OS-QNet, its computation time and complexity during inference were lower than those of OS-QNet. The computation time required by AC-OSELM to estimate the compression ratio is reduced by up to 90.0% compared to that required by OS-QNet. In particular, as \(k\) is 1 in the compression ratio estimation, the computational complexities associated with both data compression in CS and compression ratio estimation in the AC-OSELM are equal \((= O(D^2))\). Therefore, estimating the compression ratio using the AC-OSELM does not become a computational bottleneck in the data compression procedure of CS. By contrast, the computational complexity required to estimate the compression ratio for OS-QNet is greater than that required for data compression. Considering that data compression on the edge requires real-time performance in inference rather than learning, AC-OSELM-based compression ratio estimation can be considered more suitable for implementation on the edge than the OS-QNet-based counterpart.

VI. LIMITATION AND FUTURE WORK

Although OS-QNet/AC-OSELM are small-scale models, they are capable of appropriately estimating compression ratios for less complex data. However, they may be unable to appropriately learn complex data, for e.g., those that cannot be handled by CS. In such cases, it is necessary to use a large-scale model such as DRL for the compressed ratio estimation.

In learning, recovering data to calculate \(E(c)\) is a computational bottleneck, as shown in Table II. Xu et al. proposed a fast CS recovery method using a generative model based on deep learning [24]. Our future work is to realize these models on a small scale and efficiently approximate \(E(c)\) at the edge.

VII. CONCLUSION

In this study, we developed AC-OSELM as an efficient RL method for edge devices. Using AC-OSELM, the selection of actions could be performed faster than with OS-QNet. We also proposed a data compression method that uses AC-OSELM to estimate the compression ratio in CS. This allows for rapid compression ratio estimation and process execution on the edge while achieving adaptive compression for the acquired data. To evaluate the compression ratio estimation performance of the proposed AC-OSELM, it was compared with OS-QNet. The experimental results demonstrated that the proposed method outperformed the compared method with regard to the compression performance and computational cost of estimating the compression ratio. This confirms that the AC-OSELM-based compression-ratio estimation method is suitable for implementation on edge devices.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE internet of things journal, vol. 3, no. 5, pp. 637–646, 2016.
[2] D. L. Donoho, “Compressed sensing,” IEEE Transactions on information theory, vol. 52, no. 4, pp. 1289–1306, 2006.
[3] M. Sekine and S. Ikada, “Lacsle: Lightweight and adaptive compressed sensing based on deep learning for edge devices,” in 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019, pp. 1–7.
[4] V. Muih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” arXiv preprint arXiv:1312.5602, 2013.
[5] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” IEEE Transactions on neural networks, vol. 17, no. 6, pp. 1411–1423, 2006.
[6] H. Watanabe, M. Tsukada, and H. Matsutani, “An fpga-based on-device reinforcement learning approach using online sequential learning,” in 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2021, pp. 96–103.
[7] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in Advances in neural information processing systems. Citeseer, 2000, pp. 1008–1014.
[8] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in International conference on machine learning. PMLR, 2014, pp. 387–395.
[9] S. Di and F. Cappello, “Fast error-bounded lossy hpc data compression with sz,” in 2016 ieee international parallel and distributed processing symposium (ipdps). IEEE, 2016, pp. 730–739.
[10] J. Azar, A. Makhoul, M. Barhamangi, and R. Couturier, “An energy efficient iot data compression approach for edge machine learning,” Future Generation Computer Systems, vol. 96, pp. 168–175, 2019.
[11] L. Li, G. Wen, Z. Wang, and Y. Yang, “Efficient and secure image communication system based on compressed sensing for iot sensing applications,” IEEE Transactions on Multimedia, vol. 22, no. 1, pp. 82–95, 2019.
[12] S. Li, L. Da Xu, and X. Wang, “Compressed sensing signal and data acquisition in wireless sensor networks and internet of things,” IEEE Transactions on Industrial Informatics, vol. 9, no. 4, pp. 2177–2186, 2013.
[13] J. V. Frances-Villora, A. Rosado-Muñoz, M. Bataller-Monpe, J. Barrios-Aviles, and J. F. Guerrero-Martinez, “Moving learning machine towards fast real-time applications: A high-speed fpga-based implementation of the os-elm training algorithm,” Electronics, vol. 7, no. 11, p. 308, 2018.
[14] A. Papageorgiou, B. Cheng, and E. Kovacs, “Real-time data reduction at the network edge of internet-of-things systems,” in 2015 11th international conference on network and service management (CNSM). IEEE, 2015, pp. 284–291.
[15] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause, “Streaming submodular maximization: Massive data summarization on the fly,” in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 671–680.
[16] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: theory and applications,” Neurocomputing, vol. 70, no. 1-3, pp. 489–501, 2006.
[17] C. Wilson, A. Riccardi, and E. Minisci, “A novel update mechanism for q-networks based on extreme learning machines,” in 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 2020, pp. 1–7.
[18] R. Furfaro, A. Scorsoglio, R. Linares, and M. Massari, “Adaptive generalized zem-zev feedback guidance for planetary landing via a deep reinforcement learning approach,” Acta Astronautica, vol. 171, pp. 151–171, 2020.
[19] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” http://yann.lecun.com/exdb/mnist/, 2010.
[20] T. Clauwut, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, “Deep learning for classical japanese literature,” arXiv preprint arXiv:1812.01718, 2018.
[21] “Raspberry Pi 3 Model B,” Available online:https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, (accessed on 03/15/2023).
[22] S. Mitchell, M. O'Sullivan, and I. Dumling, “Pulp: a linear programming toolkit for python,” The University of Auckland, Auckland, New Zealand, vol. 65, 2011.
[23] “MacBook Air with M1 chip - Apple,” Available online:https://www.apple.com/macbook-air-m1/, (accessed on 03/15/2023).
[24] S. Xu, S. Zeng, and I. Romberg, “Fast compressive sensing recovery using generative models with structured latent variables,” in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019, pp. 2967–2971.