AN ON-DEVICE FEDERATED LEARNING APPROACH FOR COOPERATIVE ANOMALY DETECTION

A PREPRINT

Rei Ito
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
rei@arc.ics.keio.ac.jp

Mineto Tsukada
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
tsukada@arc.ics.keio.ac.jp

Hiroki Matsutani
Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
matutani@arc.ics.keio.ac.jp

July 21, 2020

ABSTRACT

Most edge AI focuses on prediction tasks on resource-limited edge devices, while the training is done at server machines, so retraining a model on the edge devices to reflect environmental changes is a complicated task. To follow such a concept drift, a neural-network based on-device learning approach is recently proposed, so that edge devices train incoming data at runtime to update their model. In this case, since a training is done at distributed edge devices, the issue is that only a limited amount of training data can be used for each edge device. To address this issue, one approach is a cooperative learning or federated learning, where edge devices exchange their trained results and update their model by using those collected from the other devices. In this paper, as an on-device learning algorithm, we focus on OS-ELM (Online Sequential Extreme Learning Machine) and combine it with autoencoder for anomaly detection. We extend it for an on-device federated learning so that edge devices can exchange their trained results and update their model by using those collected from the other edge devices. Evaluation results using a driving dataset of cars and a human activity dataset demonstrate that the proposed on-device federated learning can produce a merged model by combining trained results from multiple edge devices as accurately as a conventional backpropagation based neural network. Latency for the merging is reasonable, and it can merge the models faster than continuously executing the sequential learning.

***K*eywords** On-device learning · Federated learning · OS-ELM

1 Introduction

Most edge AI focuses on prediction tasks on resource-limited edge devices assuming that their prediction model has been trained at server machines beforehand. In this case, retraining or customizing a model for a given edge device later to reflect environmental changes is a complicated task, because the server machine needs to collect training data from the edge device, train a new model based on the collected data, and then deliver the new model to the edge device.

To realize training tasks at resource-limited edge devices, a neural-network based on-device learning approach is proposed in [1, 2], so that edge devices can train or correct their model based on incoming data at runtime. Its low-cost hardware implementation is also introduced in [2]. In this case, since a training is done independently at distributed edge devices, the issue is that only a limited amount of training data can be used for each edge device. To address this issue, one approach is a cooperative model update, where edge devices exchange their trained results and update their

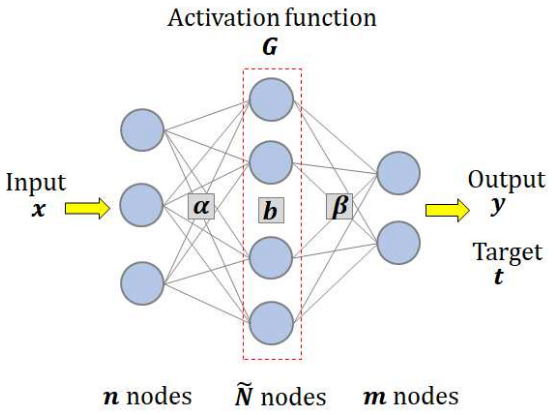

Figure 1: Single hidden-layer feedforward network (SLFN).

model using those collected from the other devices. Please note that edge devices share an intermediate form of their weight parameters instead of raw data, which is sometimes privacy sensitive.

In this paper, we use the on-device learning algorithm [1, 2], which is based on OS-ELM (Online Sequential Extreme Learning Machine) [3] for sequential training for single hidden-layer neural networks and combined with autoencoder [4] for unsupervised anomaly detection. It is then extended for the on-device federated learning so that edge devices can exchange their trained results and update their model using those collected from the other edge devices. To do this, we employ Elastic ELM (E^{2}LM) [5] that enables distributed training of neural networks, where intermediate training results are computed by multiple machines separately and then a final model is produced by combining these intermediate results. It is applied to the OS-ELM based on-device learning approach [2] for the on-device federated learning. In the evaluations, we will demonstrate that the proposed on-device federated learning can produce a merged model by combining trained results from multiple edge devices as accurately as a conventional backpropagation based neural network. It is also evaluated in terms of performance.

The rest of this paper is organized as follows. Section 2 briefly overviews baseline technologies including ELM, E^{2}LM, and OS-ELM. Section 3 proposes a model exchange and update algorithm for the on-device federated learning. Section 4 evaluates the proposed approach using a car driving dataset and a human activity dataset. Section 5 concludes this paper.

2 Preliminaries

This section briefly introduces 1) ELM (Extreme Learning Machine), 2) E^{2}LM (Elastic Extreme Learning Machine), 3) OS-ELM (Online Sequential Extreme Learning Machine), and 4) autoencoder.

2.1 ELM

ELM [6] is a batch training algorithm for single hidden-layer feedforward networks (SLFNs). As shown in Figure 1, the network consists of input layer, hidden layer, and output layer. The numbers of their nodes are denoted as n, $\tilde{N}$, and m, respectively. Assuming an n-dimensional input chunk $\boldsymbol{x} \in \boldsymbol{R}^{k \times n}$ of batch size k is given, an m-dimensional output chunk $\boldsymbol{y} \in \boldsymbol{R}^{k \times m}$ is computed as follows.

$$\boldsymbol{y} = G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta}, \tag{1}$$

where G is an activation function, $\boldsymbol{\alpha} \in \boldsymbol{R}^{\boldsymbol{n} \times \tilde{\boldsymbol{N}}}$ is an input weight matrix between the input and hidden layers, $\boldsymbol{\beta} \in \boldsymbol{R}^{\tilde{\boldsymbol{N}} \times \boldsymbol{m}}$ is an output weight matrix between the hidden and output layers, and $\boldsymbol{b} \in \boldsymbol{R}^{\tilde{\boldsymbol{N}}}$ is a bias vector of the hidden layer.

If an SLFN model can approximate m-dimensional target chunk (i.e., teacher data) $\boldsymbol{t} \in \boldsymbol{R}^{k \times m}$ with zero error, the following equation is satisfied.

$$G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})\boldsymbol{\beta} = \boldsymbol{t} \tag{2}$$

Here, the hidden-layer matrix is defined as $\boldsymbol{H} \equiv G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})$. The optimal output weight matrix $\hat{\boldsymbol{\beta}}$ is computed as follows.

$$\hat{\boldsymbol{\beta}} = \boldsymbol{H}^{\dagger}\boldsymbol{t}, \tag{3}$$

where $\boldsymbol{H}^{\dagger}$ is a pseudo inverse matrix of $\boldsymbol{H}$, which can be computed with matrix decomposition algorithms, such as SVD (Singular Value Decomposition) and QRD (QR Decomposition).

In ELM algorithm, the input weight matrix $\boldsymbol{\alpha}$ is initialized with random values and not changed thereafter. The optimization is thus performed only for the output weight matrix $\boldsymbol{\beta}$, and so it can reduce the computation cost compared with backpropagation based neural networks that optimize both $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. In addition, the training algorithm of ELM is not iterative; it analytically computes the optimal weight matrix $\boldsymbol{\beta}$ for a given input chunk in a one-shot manner, as shown in Equation 3. It can always obtain a global optimal solution for $\boldsymbol{\beta}$, unlike a typical gradient descent method, which sometimes converges to a local optimal solution.

Please note that ELM is one of batch training algorithms for SLFNs, which means that the model is trained by using all the training data for each update. In other words, we need to retrain the whole data in order to update the model for newly-arrived training samples. This issue is addressed by E^{2}LM and OS-ELM.

2.2 E^{2}LM

E^{2}LM [5] is an extended algorithm of ELM for enabling the distributed training of SLFNs. That is, intermediate training results are computed by multiple machines separately, and then a final model is produced by combining these intermediate results.

In Equation 3, assuming that rank $\boldsymbol{H} = \tilde{N}$ and $\boldsymbol{H^T H}$ is nonsingular, the pseudo inverse matrix $\boldsymbol{H}^{\dagger}$ is decomposed as follows.

$$\boldsymbol{H}^{\dagger} = \left(\boldsymbol{H^T H}\right)^{-1}\boldsymbol{H^T} \tag{4}$$

The optimal output weight matrix $\boldsymbol{\beta}$ in Equation 3 can be computed as follows.

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{H^T H}\right)^{-1}\boldsymbol{H^T t} \tag{5}$$

Assuming the intermediate results are defined as $\boldsymbol{U} = \boldsymbol{H^T H}$ and $\boldsymbol{V} = \boldsymbol{H^T t}$, the above equation is denoted as follows.

$$\hat{\boldsymbol{\beta}} = \boldsymbol{U}^{-1}\boldsymbol{V} \tag{6}$$

Here, the hidden-layer matrix and target chunk (i.e., teacher data) for newly-arrived training dataset $\Delta\boldsymbol{x}$ are denoted as $\Delta\boldsymbol{H}$ and $\Delta\boldsymbol{t}$, respectively. The intermediate results for $\Delta\boldsymbol{x}$ are denoted as $\Delta\boldsymbol{U} = \Delta\boldsymbol{H^T}\Delta\boldsymbol{H}$ and $\Delta\boldsymbol{V} = \Delta\boldsymbol{H^T}\Delta\boldsymbol{t}$.

Similarly, the hidden-layer matrix and target chunk for updated training dataset $\boldsymbol{x'} = \boldsymbol{x} + \Delta\boldsymbol{x}$ are denoted as $\boldsymbol{H'}$ and $\boldsymbol{t'}$, respectively. The intermediate results for $\boldsymbol{x'}$ are denoted as $\boldsymbol{U'} = \boldsymbol{H'^T H'}$ and $\boldsymbol{V'} = \boldsymbol{H'^T t'}$. Then, $\boldsymbol{U'}$ and $\boldsymbol{V'}$ can be computed as follows.

$$\begin{aligned}
\boldsymbol{U'} &= \boldsymbol{H'^T H'} = \begin{bmatrix}\boldsymbol{H} \\ \Delta\boldsymbol{H}\end{bmatrix}^T \begin{bmatrix}\boldsymbol{H} \\ \Delta\boldsymbol{H}\end{bmatrix} = \boldsymbol{H^T H} + \Delta\boldsymbol{H^T}\Delta\boldsymbol{H} \\
\boldsymbol{V'} &= \boldsymbol{H'^T t'} = \begin{bmatrix}\boldsymbol{H} \\ \Delta\boldsymbol{H}\end{bmatrix}^T \begin{bmatrix}\boldsymbol{t} \\ \Delta\boldsymbol{t}\end{bmatrix} = \boldsymbol{H^T t} + \Delta\boldsymbol{H^T}\Delta\boldsymbol{t}
\end{aligned} \tag{7}$$

As a result, Equation 7 can be denoted as follows.

$$\begin{aligned}
\boldsymbol{U'} &= \boldsymbol{U} + \Delta\boldsymbol{U} \\
\boldsymbol{V'} &= \boldsymbol{V} + \Delta\boldsymbol{V}
\end{aligned} \tag{8}$$

In summary, E^{2}LM algorithm updates a model in the following steps:

1. Compute $\boldsymbol{U}$ and $\boldsymbol{V}$ for the whole training dataset $\boldsymbol{x}$,
2. Compute $\Delta\boldsymbol{U}$ and $\Delta\boldsymbol{V}$ for newly-arrived training dataset $\Delta\boldsymbol{x}$,
3. Compute $\boldsymbol{U'}$ and $\boldsymbol{V'}$ for updated training dataset $\boldsymbol{x'}$ using Equation 8, and
4. Compute the new output weight matrix $\boldsymbol{\beta}$ using Equation 6.

Please note that we can compute a pair of $\boldsymbol{U}$ and $\boldsymbol{V}$ and a pair of $\Delta\boldsymbol{U}$ and $\Delta\boldsymbol{V}$ separately. Then, we can produce $\boldsymbol{U'}$ and $\boldsymbol{V'}$ by simply adding them using Equation 8. Similar to the addition of $\boldsymbol{x}$ and $\Delta\boldsymbol{x}$, subtraction and replacement operations for $\boldsymbol{x}$ are also supported.

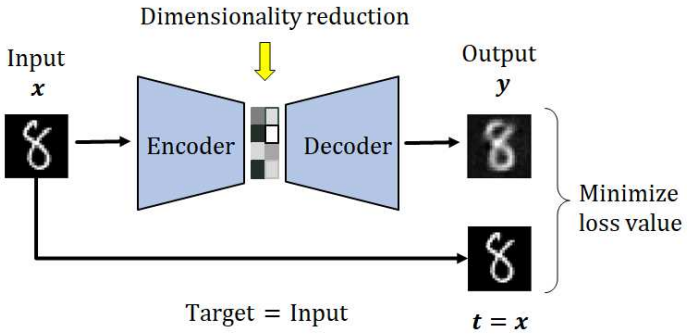

Figure 2: Autoencoder.

2.3 OS-ELM

OS-ELM [3] is an online sequential version of ELM, which can update the model sequentially using an arbitrary batch size.

Assuming that the i-th training chunk $\{\boldsymbol{x}_i \in \boldsymbol{R}^{k_i \times n}, \boldsymbol{t}_i \in \boldsymbol{R}^{k_i \times m}\}$ of batch size k_i is given, we need to compute the output weight matrix $\boldsymbol{\beta}$ that can minimize the following error.

$$\left\| \begin{bmatrix} \boldsymbol{H}_0 \\ \vdots \\ \boldsymbol{H}_i \end{bmatrix} \boldsymbol{\beta}_i - \begin{bmatrix} \boldsymbol{t}_0 \\ \vdots \\ \boldsymbol{t}_i \end{bmatrix} \right\|, \tag{9}$$

where $\boldsymbol{H}_i$ is defined as $\boldsymbol{H}_i \equiv G(\boldsymbol{x}_i \cdot \boldsymbol{\alpha} + \boldsymbol{b})$. Assuming $\boldsymbol{K_i} \equiv \begin{bmatrix} \boldsymbol{H_0} \\ \vdots \\ \boldsymbol{H_i} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H_0} \\ \vdots \\ \boldsymbol{H_i} \end{bmatrix}$ $(i \geq 0)$, the optimal output weight matrix is computed as follows.

$$\begin{aligned} \boldsymbol{\beta_i} &= \boldsymbol{\beta_{i-1}} + \boldsymbol{K_i^{-1}} \boldsymbol{H_i^T} (\boldsymbol{t_i} - \boldsymbol{H_i} \boldsymbol{\beta_{i-1}}) \\ \boldsymbol{K_i} &= \boldsymbol{K_{i-1}} + \boldsymbol{H_i^T} \boldsymbol{H_i} \end{aligned} \tag{10}$$

Assuming $\boldsymbol{P_i} \equiv \boldsymbol{K_i^{-1}}$, we can derive the following equation from Equation 10.

$$\begin{aligned} \boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \boldsymbol{P}_{i-1} \boldsymbol{H}_i^T (\boldsymbol{I} + \boldsymbol{H}_i \boldsymbol{P}_{i-1} \boldsymbol{H}_i^T)^{-1} \boldsymbol{H}_i \boldsymbol{P}_{i-1} \\ \boldsymbol{\beta}_i &= \boldsymbol{\beta}_{i-1} + \boldsymbol{P}_i \boldsymbol{H}_i^T (\boldsymbol{t}_i - \boldsymbol{H}_i \boldsymbol{\beta}_{i-1}) \end{aligned} \tag{11}$$

In particular, the initial values $\boldsymbol{P}_0$ and $\boldsymbol{\beta}_0$ are precomputed as follows.

$$\begin{aligned} \boldsymbol{P}_0 &= (\boldsymbol{H}_0^T \boldsymbol{H}_0)^{-1} \\ \boldsymbol{\beta}_0 &= \boldsymbol{P}_0 \boldsymbol{H}_0^T \boldsymbol{t}_0 \end{aligned} \tag{12}$$

As shown in Equation 11, 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 a one-shot manner; thus there is no need to retrain with all the past data unlike ELM.

In this approach, the major bottleneck is the pseudo inverse operation $(\boldsymbol{I} + \boldsymbol{H_i} \boldsymbol{P_{i-1}} \boldsymbol{H_i^T})^{-1}$. As in [1, 2], the batch size k is fixed at one in this paper so that the pseudo inverse operation of $k \times k$ matrix for the sequential training is replaced with a simple reciprocal operation; thus we can eliminate the SVD or QRD computation.

2.4 Autoencoder

Autoencoder [4] is a type of neural networks developed for dimensionality reduction, as shown in Figure 2. In this paper, OS-ELM is combined with autoencoder for unsupervised anomaly detection. In this case, the numbers of input- and output-layer nodes are the same (i.e., $n = m$), while the number of hidden-layer nodes is set to less than that of

input-layer nodes (i.e., $\tilde{N} < n$). In autoencoder, an input chunk is converted into a well-characterized dimensionally reduced form at the hidden layer. The process for the dimensionality reduction is denoted as "encoder", and that for decompressing the reduced form is denoted as "decoder". In OS-ELM, the encoding result for an input chunk $\boldsymbol{x}$ is obtained as $\boldsymbol{H} = G(\boldsymbol{x} \cdot \boldsymbol{\alpha} + \boldsymbol{b})$, and the decoding result for the hidden-layer matrix $\boldsymbol{H}$ is obtained as $\boldsymbol{y} = \boldsymbol{H} \cdot \boldsymbol{\beta}$.

In the training phase, an input chunk $\boldsymbol{x}$ is used as a target chunk $\boldsymbol{t}$. That is, the output weight matrix $\boldsymbol{\beta}$ is trained so that an input data is reconstructed as correctly as possible by autoencoder. Assuming that the model is trained with a specific input pattern, the difference between the input data and reconstructed data (denoted as loss value) becomes large when the input data is far from the trained pattern. Please note that autoencoder does not require any labeled training data for the training phase; so it is used for unsupervised anomaly detection. In this case, incoming data with high loss value should be automatically rejected before training for stable anomaly detection.

3 On-Device Federated Learning

As an on-device learning algorithm, in this paper, we employ a combination of OS-ELM and autoencoder for online sequential learning and unsupervised anomaly detection [1, 2]. It is further optimized by setting the batch size k to one, in order to eliminate the pseudo inverse operation of $k \times k$ matrix for the sequential training. A low-cost forgetting mechanism that does not require the pseudo inverse operation is also proposed in [2].

In practice, anomaly patterns should be accurately detected from multiple normal patterns. To improve the accuracy of anomaly detection in such cases, we employ multiple on-device learning instances, each of which is specialized for each normal pattern as proposed in [7]. Also, the number of the on-device learning instances can be dynamically tuned at runtime as proposed in [7].

In this paper, the on-device learning algorithm is extended for the on-device federated learning by applying the E^{2}LM approach to the OS-ELM based sequential training. In this case, edge devices can share their intermediate trained results and update their model using those collected from the other edge devices. In this section, OS-ELM algorithm is analyzed so that the E^{2}LM approach is applied to OS-ELM for enabling the cooperative model update. The proposed on-device federated learning approach is then illustrated in detail.

3.1 Modifications for OS-ELM

Here, we assume that edge devices exchange the intermediate results of their output weight matrix $\boldsymbol{\beta}$ (see Equation 6). These intermediate results are obtained by $\boldsymbol{U} = \boldsymbol{H^T H}$ and $\boldsymbol{V} = \boldsymbol{H^T t}$, based on E^{2}LM algorithm. Please note that the original E^{2}LM approach is designed for ELM, which assumes a batch training, not a sequential training. That is, $\boldsymbol{U}$ and $\boldsymbol{V}$ are computed by using the whole training dataset. On the other hand, our on-device learning algorithm relies on the OS-ELM based sequential training, in which the weight matrix is sequentially updated every time a new data comes. If the original E^{2}LM approach is directly applied to our on-device learning algorithm, all the past dataset must be preserved in edge devices, which would be infeasible for resource-limited edge devices.

To address this issue, OS-ELM is analyzed as follows. In Equation 10, $\boldsymbol{K_i}$ is defined as $\boldsymbol{K_i} \equiv \begin{bmatrix} \boldsymbol{H_0} \\ \vdots \\ \boldsymbol{H_i} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H_0} \\ \vdots \\ \boldsymbol{H_i} \end{bmatrix}$ $(i \geq 0)$, which indicates that it accumulates all the hidden-layer matrixes that have been computed with up to the $\boldsymbol{i}$-th training chunk. In this case, $\boldsymbol{U}$ and $\boldsymbol{V}$ of E^{2}LM can be computed based on $\boldsymbol{K_i}$ and its inverse matrix $\boldsymbol{P_i}$ of OS-ELM as follows.

$$\begin{aligned} \boldsymbol{U_i} &= \boldsymbol{K_i} = \boldsymbol{P_i}^{-1} \\ \boldsymbol{V_i} &= \boldsymbol{U_i}\boldsymbol{\beta_i}, \end{aligned} \tag{13}$$

where $\boldsymbol{U_i}$ and $\boldsymbol{V_i}$ are intermediate results for the $\boldsymbol{i}$-th training chunk. $\boldsymbol{U_i}$ and $\boldsymbol{V_i}$ can be sequentially computed with the previous training results. To support the on-device federated learning, Equation 13 is newly inserted to the sequential training algorithm of OS-ELM, which will be introduced in Section 3.2.

3.2 Cooperative Mode Update Algorithm

Figure 3 illustrates a cooperative model update of the proposed on-device federated learning. It consists of the following three phases:

1. Sequential training on edge devices,
2. Exchanging their intermediate results via a cloud server, and

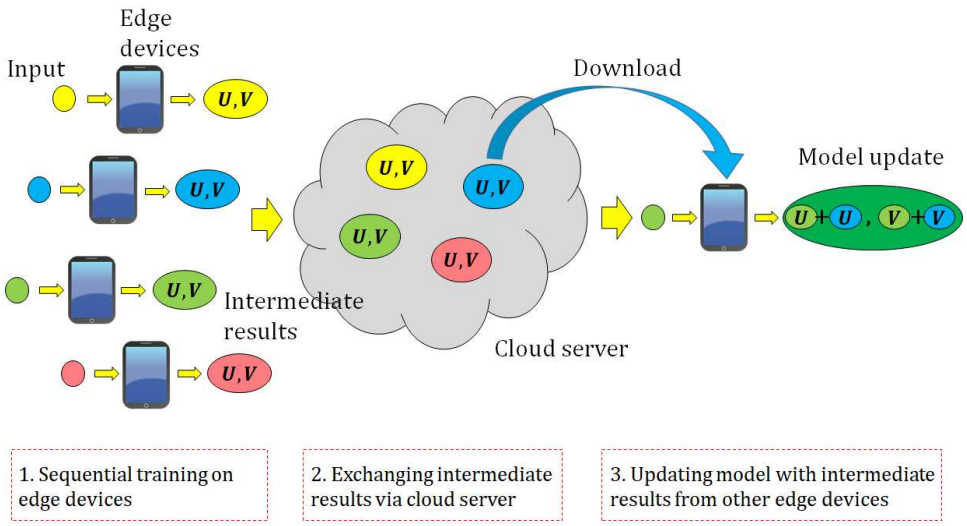

Figure 3: Overview of cooperative model update.

3. Updating their model with necessary intermediate results from the other edge devices.

First, edge devices independently execute a sequential training by using OS-ELM algorithm. They also compute the intermediate results $\boldsymbol{U}$ and $\boldsymbol{V}$ by Equation 13. Second, they upload their intermediate results to a cloud server. We assume that the input weight matrix $\boldsymbol{\alpha}$ and the bias vector $\boldsymbol{b}$ are the same in the edge devices. They download necessary intermediate results from the cloud server if needed. They update their model based on their own intermediate results and those downloaded from the cloud server by Equation 8.

Figure 4 shows a flowchart of the proposed cooperative model update between two devices: Device-A and Device-B. Assuming Device-A sends its intermediate results and Device-B receives them for updating its model, their cooperative model update is performed by the following steps.

1. Device-A and Device-B sequentially train their own model by using OS-ELM algorithm. In other words, they compute the output weight matrix $\boldsymbol{\beta}$ and its intermediate result $\boldsymbol{P}$ by Equation 11.
2. Device-A computes the intermediate results $\boldsymbol{U_A}$ and $\boldsymbol{V_A}$ by Equation 13 to share them with other edge devices. Device-B also computes $\boldsymbol{U_B}$ and $\boldsymbol{V_B}$. They upload these results to a cloud server.
3. Assuming Device-B demands the Device-A's trained results, it downloads $\boldsymbol{U_A}$ and $\boldsymbol{V_A}$ from the cloud server.
4. Device-B integrates their intermediate results by computing $\boldsymbol{U_B} \leftarrow \boldsymbol{U_A} + \boldsymbol{U_B}$ and $\boldsymbol{V_B} \leftarrow \boldsymbol{V_A} + \boldsymbol{V_B}$ by using E^{2}LM algorithm.
5. Device-B updates $\boldsymbol{P_B}$ and $\boldsymbol{\beta_B}$ by computing $\boldsymbol{P_B} \leftarrow \boldsymbol{U_B}^{-1}$ and $\boldsymbol{\beta_B} \leftarrow \boldsymbol{U_B}^{-1}\boldsymbol{V_B}$.
6. Device-B can execute prediction and/or sequential training of OS-ELM algorithm by using the integrated $\boldsymbol{P_B}$ and $\boldsymbol{\beta_B}$.

Edge devices can share their trained results by exchanging their intermediate results $\boldsymbol{U}$ and $\boldsymbol{V}$ in the proposed on-device federated learning approach, which can mitigate the privacy issues since they do not share raw data for the cooperative model update. Please note that the intermediate results $\boldsymbol{U}$ and $\boldsymbol{V}$ in Equation 13 should be updated only when they are sent to a cloud server or the other edge devices; so there is no need to update them for every input chunk. Alternatively, it is possible to compute $\boldsymbol{U}$ and $\boldsymbol{V}$ at the server, not edge devices, though they need to upload raw data to the server.

4 Evaluations

The behavior of the proposed on-device federated learning approach is demonstrated by combining trained results from multiple edge devices. Prediction results using the combined model are compared to those produced by a conventional 3-layer BP-NN (backpropagation based neural network). In addition, the proposed on-device federated learning is evaluated in terms of the model merging latency, and it is compared to a conventional sequential learning approach. Specification of the experimental machine is shown in Table 1.

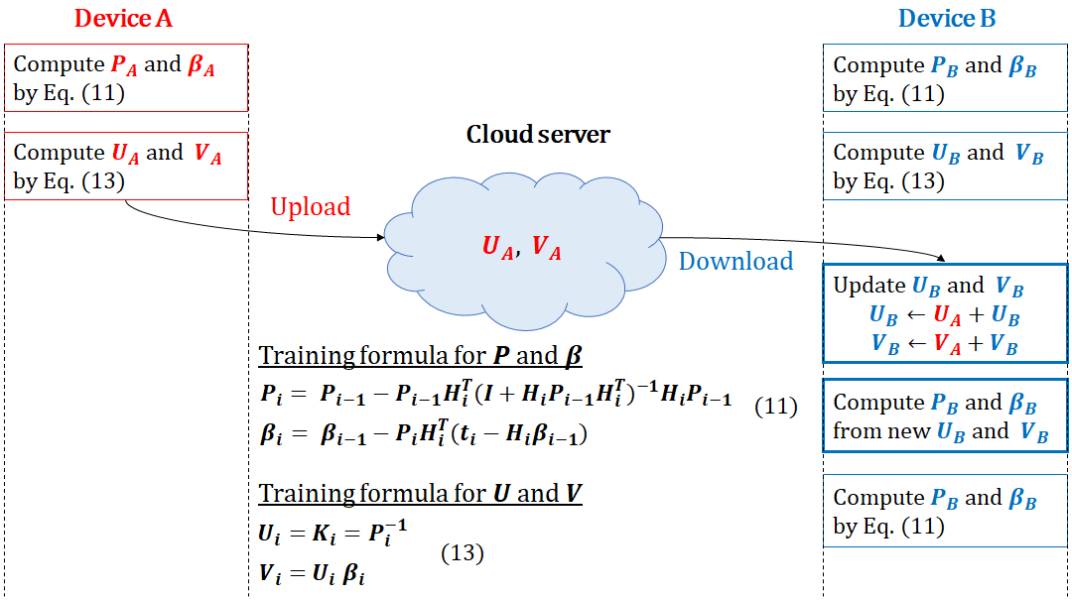

Figure 4: Flowchart of cooperative model update.

4.1 Evaluation Environment

4.1.1 Datasets

The evaluations are conducted with two datasets shown in Table 2. UAH-DriveSet [8] contains car driving histories of six drivers simulating three different driving patterns: aggressive, drowsy, and normal. It can be used for the aggressive driving detection. Their car speed was extracted from the GPS data obtained from a smartphone fixed in their cars. The sampling frequency of the car speed was 1Hz. In our experiment, the car speed is quantized with 15 levels (1 level = 10km/h). Assuming a state is assigned to each speed level, we can build a state-transition probability table with 15×15 entries, each of which represents a probability of a state transition from one state to another.

Smartphone HAR dataset [9] contains human activity recordings of 30 volunteers simulating six activities: walking, walking_upstairs, walking_downstairs, sitting, standing, and laying. It can be used for human activity recognition. Their 3-axial linear acceleration and 3-axial angular velocity were captured at a constant rate of 50Hz using waist-mounted smartphone with embedded inertial sensors. The sensor signals were pre-processed. As a result, 561 features consisting of time and frequency domain variables were obtained for each time-window.

4.1.2 Setup

A vector of 225 features from the car driving dataset and that of 561 features from the human activity dataset are fed to the neural-network based on-device learning algorithm [1, 2] for anomaly detection. The numbers of input-layer nodes n, hidden-layer nodes $\tilde{N}$, and output-layer nodes m are 225, 16, and 225, respectively, in the experiment with the car driving dataset. Whereas, they are 561, 128, and 561, respectively, in the experiment with the human activity dataset. The activation function G is the sigmoid, and the loss function L is the mean square error. The forget factor α is 1 (i.e., no forgetting) [2]. The batch size k is fixed to 1 [1]. The number of training epochs is 1. The number of anomaly detection instances is 2 [7]. These instances are denoted as Device-A and Device-B in the evaluations. The OS-ELM based anomaly detection with the proposed cooperative model update is implemented with Python 3.6.4 and NumPy 1.14.1.

Table 1: Specification of experimental machine.

OS	Ubuntu 17.10
CPU	Intel Core i5 7500 3.4GHz
DRAM	8GB
Storage	SSD 480GB

Table 2: Two datasets.

Name	Features	Classes
UAH-DriveSet [8]	225	3
Smartphone HAR [9]	561	6

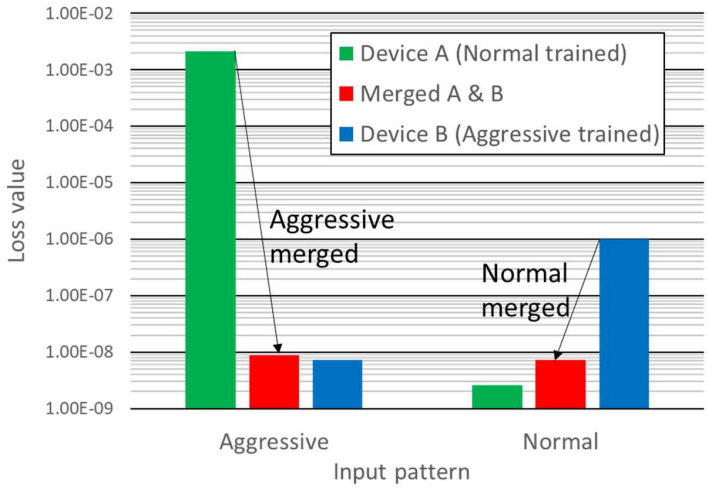

Figure 5: Loss values before and after cooperative model update with car driving dataset.

As a comparison with the proposed OS-ELM based anomaly detection, a 3-layer BP-NN based autoencoder is implemented with TensorFlow v1.12.0 [10]. In the BP-NN based autoencoder, the batch size is 64 and the number of training epochs is 10 unless otherwise stated in order to achieve the best generalization capability.

4.2 Loss Values Before and After Model Update

Here, we compare loss values before and after the cooperative model update. Below is the experiment scenario using the two instances with the car driving dataset.

1. Device-A trains its model so that the normal driving pattern becomes normal. Device-B trains its model so that the aggressive driving pattern becomes normal.
2. Aggressive and normal driving patterns are fed to Device-A to evaluate the loss values.
3. Device-B uploads its intermediate results to a cloud server, and Device-A downloads them from the cloud server.
4. Device-A updates its model based on its own intermediate results and those from Device-B.
5. Aggressive and normal driving patterns are fed to Device-A to evaluate the loss values.

Below is the experiment scenario using the human activity dataset.

1. Device-A trains its model so that the sitting pattern becomes normal. Device-B trains its model so that the laying pattern becomes normal.
2. Walking, walking_upstairs, walking_downstairs, sitting, standing, and laying patterns are fed to Device-B to evaluate the loss values.
3. Device-A uploads its intermediate results to a cloud server, and Device-B downloads them from the cloud server.
4. Device-B updates its model based on its own intermediate results and those from Device-A.
5. Walking, walking_upstairs, walking_downstairs, sitting, standing, and laying patterns are fed to Device-B to evaluate the loss values.

In these scenarios, the loss values after Step 2 are denoted as “before the cooperative model update”. Those after Step 5 are denoted as “after the cooperative model update”. In this setup, after the cooperative model update, Device-A that has merged Device-B and Device-B that has merged Device-A are identical. A low loss value means that a given input pattern is well reconstructed by autoencoder, which means that the input pattern is normal in the edge device. In the first scenario, Device-A is adapted to aggressive and normal driving patterns with the car driving dataset. In the second one, Device-B is adapted to sitting and laying patterns with the human activity dataset.

Figure 5 shows the loss values before and after the cooperative model update with the car driving dataset. X-axis represents the input patterns. Y-axis represents the loss values in a logarithmic scale. Green bars represent loss values

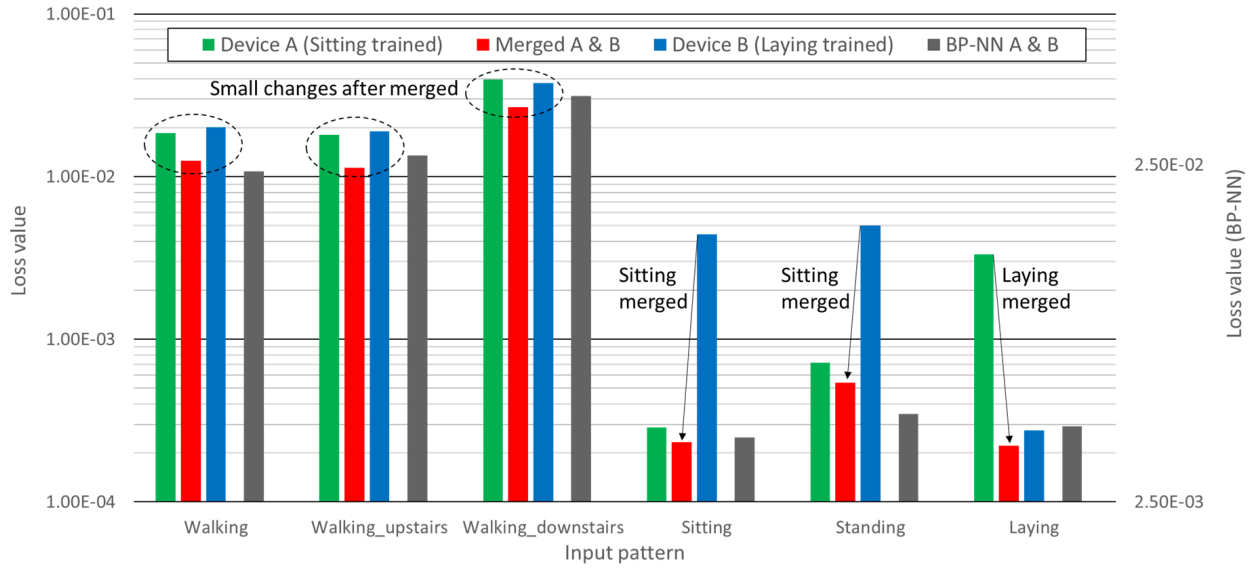

Figure 6: Loss values before and after cooperative model update with human activity dataset.

of Device-A before the cooperative model update, while red bars represent those after the cooperative model update. Blue bars represent loss values of Device-B. In the case of aggressive pattern, the loss value of Device-A before the cooperative model update (green bar) is high, because Device-A is trained with the normal pattern. The loss value then becomes quite low after integrating the intermediate results of Device-B to Device-A (red bar). This means that the trained result of Device-B is correctly adapted to Device-A. In the case of normal pattern, the loss value before merging (green bar) is low, but it slightly increases after the trained result of Device-B is merged (red bar). Nevertheless, the loss value is still quite low. We can observe the same tendency for Device-B by comparing the blue and red bars.

Figure 6 shows the loss values before and after the cooperative model update with the human activity dataset. Regarding the loss values, the same tendency with the driving dataset can be observed. In the case of sitting pattern, the loss value of Device-B before the cooperative model update (blue bar) is high, because Device-B is trained with the laying pattern. Then, the loss value becomes low after the trained result of Device-A is merged (red bar). In the case of laying pattern, the loss value of Device-A before merging (green bar) is high and significantly decreased after merging of the trained result of Device-B (red bar). On the other hand, in the walking, walking_upstairs, and walking_downstairs patterns, their loss values before and after the cooperative model update are relatively close. These input patterns are detected as abnormal even after the cooperative model update, because they are not normal for both Device-A and Device-B. In the case of standing pattern, the similar tendency as the sitting pattern is observed. The loss value becomes low after the trained result of Device-A is merged to Device-B. This means that there is a similarity between the sitting pattern and standing pattern.

As a counterpart of the proposed OS-ELM based anomaly detection, a 3-layer BP-NN based autoencoder is implemented. The BP-NN based autoencoder is trained with the sitting pattern and laying pattern. In Figure 6, black bars (Y-axis on the right side) represent loss values by BP-NN based autoencoder in a logarithmic scale. Please note that absolute values of its loss values are different from OS-ELM based ones since their training algorithms are different. Nevertheless, the tendency of the BP-NN based autoencoder (black bars) are very similar to that of the proposed cooperative model update (red bars). This means that model of Device-B after the trained result of Device-A is merged can distinguish between normal and abnormal in these input patterns as accurately as the BP-NN based autoencoder.

4.3 Training, Prediction, and Merging Latencies

In this section, the proposed on-device federated learning is evaluated in terms of training, prediction, and merging latencies with the human activity dataset. Batch size of BP-NN and the number of training epochs are set to 1 for a fair comparison with the proposed OS-ELM based anomaly detection. They are compared in terms of the following latencies.

- "Training latency" is an elapsed time from receiving an input sample until the parameter is optimized by using OS-ELM or BP-NN.

Table 3: Training, prediction, and merging latencies [msec]

Number of hidden-layer nodes $\tilde{N} = 64$			
	Training latency	Prediction latency	Merging latency
OS-ELM	**0.471**	0.089	**5.78**
BP-NN	0.588	0.290	N/A
Number of hidden-layer nodes $\tilde{N} = 128$			
	Training latency	Prediction latency	Merging latency
OS-ELM	**0.794**	0.106	**21.8**
BP-NN	0.980	0.364	N/A

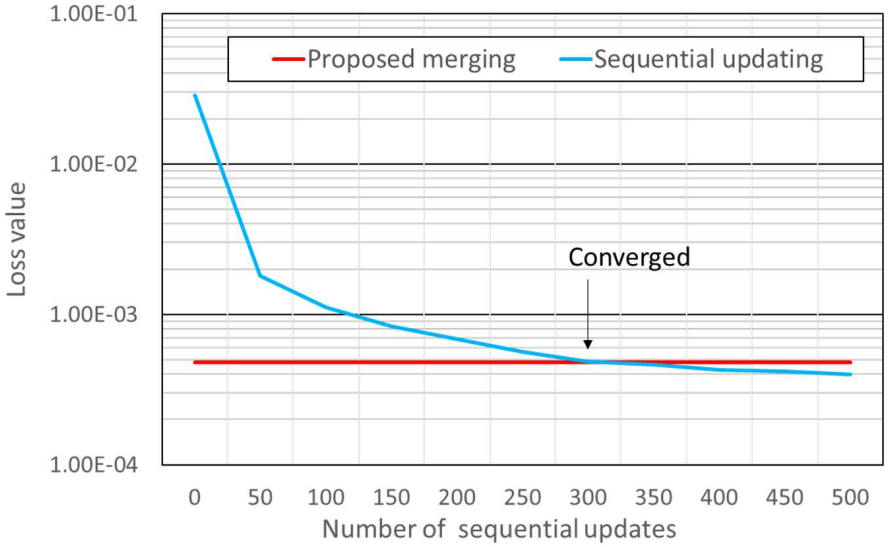

Figure 7: Convergence of loss values of merging and sequential updating.

- "Prediction latency" is an elapsed time from receiving an input sample until its loss value is computed by using OS-ELM or BP-NN.
- "Merging latency" is an elapsed time from receiving intermediate results $\boldsymbol{U}$ and $\boldsymbol{V}$ until a model update with the intermediate results is finished.

These latencies are measured on the experimental machine shown in Table 1.

Table 3 shows the evaluation results in the cases of $\tilde{N} = 64$ and $\tilde{N} = 128$. The number of input features is 561. The merging latency is larger than those of training and prediction latencies, and it depends on the number of hidden-layer nodes because of the inverse operations of $\tilde{N} \times \tilde{N}$ (size of matrix $\boldsymbol{U}$ is $\tilde{N} \times \tilde{N}$). Thus, it is 5.78 msec when $\tilde{N} = 64$, while it becomes 21.8 msec when $\tilde{N} = 128$. These latencies are still low.

4.4 Convergence of Loss Values

The proposed cooperative model update can merge trained results of different input patterns at a time. On the other hand, the original OS-ELM can intrinsically adapt to new normal patterns by continuously executing the sequential training of the new patterns. These two approaches (i.e., the proposed merging and the conventional sequential updating) are evaluated in terms of convergence of loss values for a new normal pattern. The number of hidden-layer nodes $\tilde{N}$ is 128.

In this experiment, Device-A trains its model so that the laying pattern becomes normal, and Device-B trains its model so that the walking pattern becomes normal. In the proposed merging, the trained result of Device-A is integrated to Device-B so that the laying pattern becomes normal in Device-B. In the case of the conventional sequential updating, Device-B continuously executes sequential training of the laying pattern, so that the loss value of the laying pattern is gradually decreased. Its decreasing loss value is evaluated at every 50 sequential updates and compared to that of the proposed merging.

Figure 7 shows the results. X-axis represents the number of sequential updates in the conventional sequential updating. Y-axis represents loss values of the laying pattern in a logarithmic scale. Red line represents the loss value of Device-B after the proposed merging; thus, the loss value is low and constant. Blue line represents the loss value of Device-B when sequentially updating its model by the laying pattern; thus, the loss value is decreased as the number of sequential updates increases. Then, the loss value becomes as low as that of the merged one (red line) when the number of sequential updates is approximately 300. For 300 sequential updates, at least 0.794×300 msec is required for the convergence, while the proposed cooperative model update (i.e., merging) requires only 21.8 msec. Thus, the proposed cooperative model update is useful to merge the trained results of the other edge devices.

5 Conclusions

In this paper, we focused on a neural-network based on-device learning approach so that edge devices can train or correct their model based on incoming data at runtime in order to adapt to a given environment. Since a training is done independently at distributed edge devices, the issue is that only a limited amount of training data can be used for each edge device. To address this issue, in this paper, the on-device learning algorithm was extended for the on-device federated learning by applying the E^2LM approach to the OS-ELM based sequential training. In this case, edge devices can share their intermediate trained results and update their model using those collected from the other edge devices. We illustrated an algorithm for the proposed cooperative model update. Evaluation results using a driving dataset of cars and a human activity dataset demonstrated that the proposed cooperative model update can produce a merged model by combining trained results from the other edge devices as accurately as a conventional 3-layer BP-NN based autoencoder. A latency for merging two models is 21.8 msec when the numbers of input- and hidden-layer nodes are 561 and 128, respectively. It can merge trained models faster than continuously executing the sequential learning.

References

[1] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. OS-ELM-FPGA: An FPGA-Based Online Sequential Unsupervised Anomaly Detector. In *Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par'18) Workshops*, pages 518–529, August 2018.

[2] Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. A Neural Network-Based On-device Learning Anomaly Detector for Edge Devices. *IEEE Transactions on Computers (TC)*, 69(7):1027–1044, July 2020.

[3] Nan-Ying Liang, Guang-Bin Huang, P. Saratchandran, and N. Sundararajan. Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, November 2006.

[4] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[5] Junchang Xin, Zhiqiong Wang, Luxuan Qu, and Guoren Wang. Elastic Extreme Learning Machine for Big Data Classification. *Neurocomputing*, 149:464–471, February 2015.

[6] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'04)*, pages 985–990, July 2004.

[7] Rei Ito, Mineto Tsukada, Masaaki Kondo, and Hiroki Matsutani. An Adaptive Abnormal Behavior Detection using Online Sequential Learning. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing (EUC'19)*, pages 436–440, August 2019.

[8] The UAH-DriveSet. `http://www.robesafe.com/personal/eduardo.romera/uah-driveset/`.

[9] Human Activity Recognition Using Smartphones Data Set. `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`.

[10] TensorFlow. `https://www.tensorflow.org/`.