https://doi.org/10.20965/jaciii.2017.p0856

Paper:

XCSR Learning from Compressed Data Acquired by Deep Neural Network

Kazuma Matsumoto*, Takato Tatsumi*, Hiroyuki Sato*, Tim Kovacs, and Keiki Takadama***

*The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
E-mail: {kazuma.m@, tatsumi@, sato@hc., keiki@inf.}uec.ac.jp
**The University of Bristol
MVB, Woodland Rd., Bristol, BS8 1UB, United Kingdom
E-mail: tim.kovacs@bristol.ac.uk
[Received March 21, 2017; accepted July 21, 2017]

The correctness rate of classification of neural networks is improved by deep learning, which is machine learning of neural networks, and its accuracy is higher than the human brain in some fields. This paper proposes the hybrid system of the neural network and the Learning Classifier System (LCS). LCS is evolutionary rule-based machine learning using reinforcement learning. To increase the correctness rate of classification, we combine the neural network and the LCS. This paper conducted benchmark experiments to verify the proposed system. The experiment revealed that: 1) the correctness rate of classification of the proposed system is higher than the conventional LCS (XCSR) and normal neural network; and 2) the covering mechanism of XCSR raises the correctness rate of proposed system.

Keywords: LCS, XCS, XCSR, neural network, deep learning

1. Introduction

Neural networks are one of the classification systems attracting a lot of attention these days because of the ongoing advances in the technology of deep learning [1], which is a machine learning mechanism of neural networks. The correctness rate of classification by neural networks becomes higher using deep learning and the accuracy of neural networks may even exceed that of the human brain in some instances.

This paper proposes a hybrid system of the neural network and Learning Classifier System (LCS) [2]. LCS is an evolutionary rule-based machine learning method that applies reinforcement-based learning. A neural network needs to learn intuitively when it receives data that cannot be classified using the models learned so far. However, LCS has the potential to accurately classify such data immediately without needing to learn. For example, when newly input data needs to be appropriately classified even though it is close to the outliers, it is necessary for the neural network to perform additional learning for such outlier data. Furthermore, the neural network may not be able to learn the correct classification depending on the data. However, LCS can classify the data by using multiple classifiers, including classifiers that specialize in outlier-like data. To overcome the problem of data classification in neural networks, this paper proposes a system which is a combination of the LCS method and the inherent capabilities of neural networks.

In detail, we use a deep neural network and XCSR [3] which is an extension of LCS for continuous real numbers. The neural network extracts the features of the environmental input, and inputs these features into the XCSR to learn. Finally, the XCSR classifies the inputs from the extracted features. The purposed system of this paper goes on to verify the proposed system using benchmarking problems.

In the recent past, hybrid methods of neural networks and other machine learning methods have been proposed to expand the possibility of neural networks. NeuroEvolution of Augmenting Topologies (NEAT) [4] is one such hybrid system of genetic algorithms and neural networks, which improves performance by adjusting the parameters of the neural network (related to the layer architecture and the number of notes) using a genetic algorithm. Deep Q-Network (DQN) [5] is another hybrid system of Q-learning (a reinforcement learning method [6]) and neural networks. DQN extracts the features from the input images through the neural network and selects the best course of action by Q-learning. Although NEAT, DQN, and the proposed system are hybrid methods of the neural network and other machine learning methods, NEAT and DQN are unable to classify the low frequency outlier-like input data without requiring the additional learning for such an input.

This paper is organized as follows: Sections 2 and 3 describe the mechanism of the deep neural network and the XCSR, respectively. In Section 4, we explain the detailed mechanism of the proposed system. Section 5 describes the benchmark classification experiment and we provide the conclusions of this paper in Section 6.

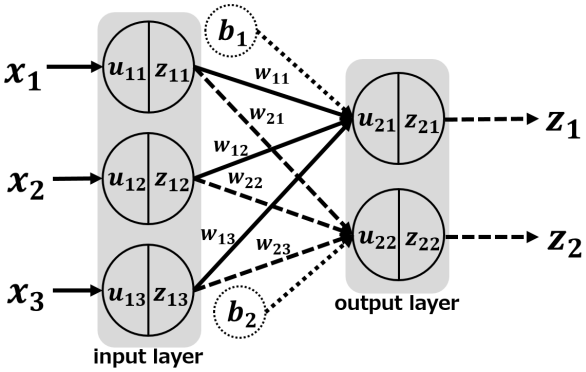


Fig. 1. Example of a neural network.

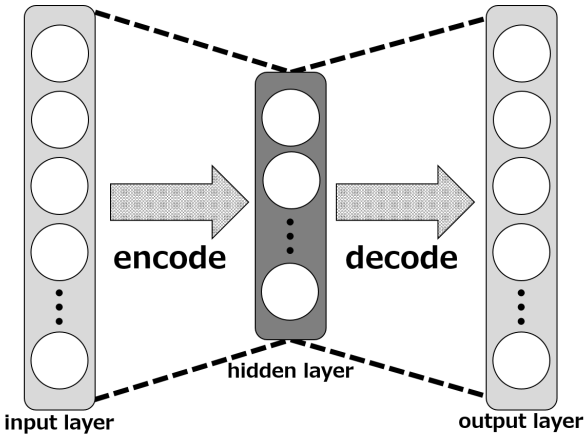


Fig. 2. Overview of an autoencoder.

2. Deep Learning

As the proposed system employs both the neural network and the autoencoder, they are explained in this section for better understanding of the proposed system.

Deep learning is the machine learning of deep neural networks. In this section, we explain basic neural networks, deep neural networks for classification (Deep Learning) , autoencoders [7], which denote a specific architecture of the neural network.

2.1. Neural Network

A neural network is a classifier modeled on the human brain [8]. A neural network is composed of nodes, layers and their interconnections.

1. An *input layer* is the first set of nodes sensing an environmental input $\vec{x} = \{x_1, x_2, \ldots, x_n\}$, where n is a length of input.
2. A *hidden layer* is a set of nodes that converts the sensed input into output. Each hidden layer is connected to the neighboring layers, i.e., the input layer, other hidden layers, or the output layer. Each hidden layer represents an internal expression of input inherited from the expression of the previous layer. While neural networks can have no hidden layers (the input layer is directly connected to the output layer) as shown in **Fig. 1**, typical neural networks have one or more hidden layers to apply complex problems.
3. The *output layer* is the layer located at the end of a network, which outputs the class or values.

As shown in **Fig. 1**, each node in a layer is connected to another node in the next layer with a *weight* value w. In **Fig. 1**, the parameter b denotes a bias, z is the output of each node, and u is the summation of the given values from the previous layer. The neural network aims to learn the values of the parameter w, which is initially set to a random value based on the Gaussian distribution. The summation u is obtained by Eq. (1), where m denotes the number of nodes in the previous layer.

$$u_{i,k} = b_k + \sum_{j=1}^{m} w_{k,j} z_{k,j} \quad \ldots\ldots\ldots\ldots \quad (1)$$

For the nodes connected to the input layer, $u_{1,j}$ can be the corresponding input's element x_j. Then, the output $z_{i,k}$ of each node is calculated using an activation function as $z_{i,k} = f(u_{i,k})$. The activation function can be set differently for each layer.

2.2. Autoencoder

An autoencoder is an algorithm used for dimensional reduction using a neural network. The autoencoder works by reproducing the input through the output. The learned hidden layers can compress the input, which represents the features of input. As the answer is the input itself, the autoencoder indicates unsupervised learning. The autoencoder learns the weight values of each node to minimize the error or the difference between the input and the output, when the neural network consists of three layers. **Fig. 2** shows an overview of an autoencoder. Specifically, to reproduce the input through the output layer, the number in nodes of the output layer is set to the same value as the input length. When a hidden layer detects a smaller number of nodes than the input length, it encodes a sensed input to a low-dimensional input. The first half of the input layer performs the role of encoding (compressing) the input to obtain the features of the input, and the second half layer performs decoding of the encoded input to its original representation. The proposed system uses this compression function of the first half layers of the autoencoder as a compressor. For the input and output layers, the activation function denoted by Eq. (2) is employed; while for the hidden layer, the logistic sigmoid function (Eq. (3)) is often used as the activation function.

$$f(u_{i,k}) = u_{i,k} \quad \ldots\ldots\ldots\ldots \quad (2)$$

$$f(u_{i,k}) = \frac{1}{1+e^{-u_{i,k}}} \quad \ldots\ldots\ldots\ldots \quad (3)$$

For the measurement of discrepancy between the input and the output (i.e., the decoded input), the following error function E is employed:

$$E = \frac{1}{2} \sum_{n=1}^{N} ||\vec{x}_n - \vec{y}(\vec{x}_n)||^2 \quad \ldots\ldots\ldots\ldots \quad (4)$$

where, $\vec{y}(\vec{x}_n)$ is the output decoded by the hidden layer with the input $\vec{x}_n$. Then, to minimize the error E, backpropagation [9] updates the weights w and biases b with a parameter δ. The parameter δ is calculated by Eq. (5).

$$\delta_{i,k} = \sum_{j=1}^{m} \delta_{i+1,j}(w_{i+1,j} f'(u_{k,j})) \quad \ldots\ldots (5)$$

Here, δ in the output layer's node is calculated by Eq. (6).

$$\delta_{L,i} = y_i - x_i \quad \ldots\ldots\ldots\ldots (6)$$

where x is the input and y is the output of the autoencoder.

Using δ, the amount of weight-update Δw is calculated as follows;

$$\Delta w_{i,k} = \delta_{i,k} z_{i-1,k} \quad \ldots\ldots\ldots\ldots (7)$$

Finally, we can update weights w using Eq. (8).

$$w_{i,k} \leftarrow w_{i,k} - \varepsilon \Delta w_{i,k} \quad \ldots\ldots\ldots\ldots (8)$$

Here, the parameter ε is a learning rate, which controls how much the weight is increased or decreased for each update. In summary, the autoencoder performs the following steps:

1. Given $\vec{x}$ to input layer.
2. Output $\vec{y}$ through hidden layer.
3. Calculate δ of output layer using $\vec{x}$ and $\vec{y}$ (Eq. (6)).
4. Calculate all δ without output layer's from output layer to input layer (Eq. (5)).
5. Calculate all Δw (Eq. (7)).
6. Update all weight w using Δw (Eq. (8)).
7. Return to step 1. (Use the next training sample.)

The autoencoder that has more than five layers is called a deep autoencoder. As the layers increase, potential of representation increases and decoding errors can be reduced.

2.3. Deep Neural Network for Classification

To use a neural network as a classifier, the softmax function (Eq. (9)) is employed as an activation function for the output layer.

$$f(u_{i,k}) = \frac{e^{u_{i,k}}}{\sum_{l=1}^{N} e^{u_{i,l}}} \quad \ldots\ldots\ldots\ldots (9)$$

The length of the input layer's nodes is the input size, and the length of the output layer's nodes denotes the number of classes. The output layer outputs a possibility of belonging to each class in a range of 0 to 1 and the classes are tagged to the nodes, respectively. The basic learning algorithm (backpropagation) is the same as the autoencoder. However, classifier networks indicate supervised learning and we need to give a class possibility as an answer to the output layer with 1-of-K expression. The error function of neural networks for classification is shown as Eq. (10), where K is a number of classes, and $\vec{d}$ is a label which is expressed with 1-of-K representation.

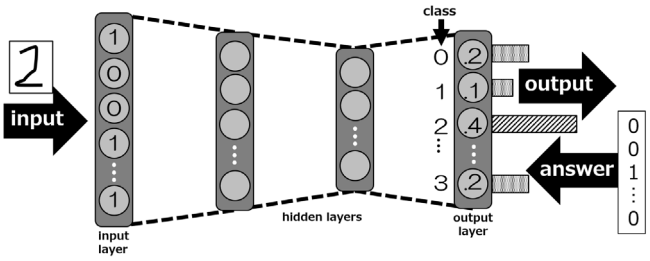


Fig. 3. An example of a deep neural network for classification (handwriting number recognition).

$$E = -\sum_{k=1}^{K} \vec{d_{nk}} \log \vec{y_k}(\vec{x_k}) \quad \ldots\ldots\ldots (10)$$

Figure 3 shows an example of a deep neural network used for classification with handwriting number recognition problem. A neural network that has more than two hidden layers such as one shown in **Fig. 3** is called a deep neural network. To prevent them from getting stuck in localized solutions, deep neural network weights are initialized by stacking autoencoders [1].

Deep neural networks can learn using a higher representation than the normal neural networks, and it makes the classification accuracy of deep neural networks more accurate. As neural networks adjust weight parameters to minimize errors in all samples, it is difficult to learn low frequency outlier-like inputs.

3. XCSR

The XCSR classifier system is an extension of the XCS classifier system [10] with a continuous real-valued coding for classifiers. In this section, after we explain the framework of the XCS, we describe the modifications made to the XCSR when compared with the XCS. **Fig. 4** shows the architecture of the XCS.

XCS is one of the major LCS for Boolean input. The LCS is an evolutionary rule-based machine learning mechanism. Because the LCS acquires knowledge with generalized if-then rules called classifiers, it is easy for humans to understand the obtained knowledge in the LCS.

3.1. Classifier

In the XCS, classifiers consist of a condition, an action, and five main parameters: (i) the prediction p, which estimates the average payoff that the system expects when the classifier is used; (ii) the prediction error ε, which estimates the average absolute error of the prediction p; (iii) the fitness F, which estimates the average relative accuracy of the payoff prediction given by p; (iv) the action set size as, which estimates the average sizes of the action sets this classifier has belonged to; and finally (v) the

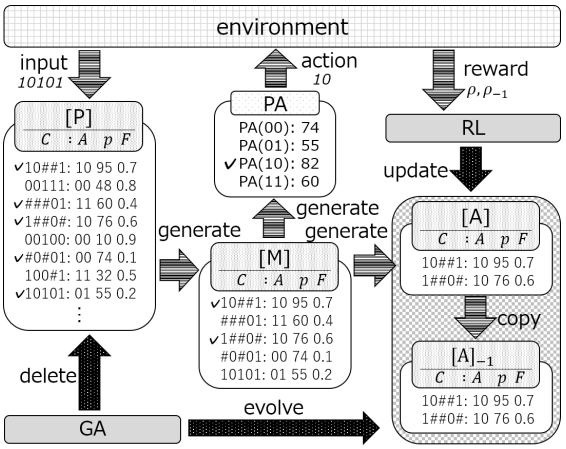


Fig. 4. The XCS architecture.

numerosity *num*, which indicates how many copies of the classifiers with the same condition and the same action are present in the population.

3.2. XCS Mechanism

The XCS is a reinforcement learning method in which generalization is obtained through the evolution of a population of condition-action-prediction rules (called classifiers). A detailed algorithmic description can be found in [11]. The XCS is composed of performance, reinforcement, discovery components, and subsumption operation.

3.2.1. Performance Component

At each time step, the XCS builds a match set $[M]$ containing the classifiers in the population $[P]$, whose condition matches the current sensory inputs; if $[M]$ does not contain all the possible actions, the covering operation takes place and creates a set of classifiers that match and cover all the missing actions. The covering operation is activated when the match set contains less than θ_{mna} actions; however, θ_{mna} is always set to the number of available actions so that the match set includes all the actions. This process ensures that the XCS can evolve a complete mapping so that it can predict the effect of every possible action in any state in terms of the expected returns. This implies that the XCS can generate classifiers that match all input, including the low-frequency outlier-like input, by the covering operation. However, because the XCS needs to create all patterns of condition-action rules, it cannot solve high-dimensional problems easily. For each possible action a_i in $[M]$, the XCS computes the system prediction $P(s_t, a_i)$, which estimates the payoff that the XCS expects if the action a_i is performed at a state s_t. The system prediction is computed as the fitness weighted average of the predictions of the classifiers in $[M]$, $cl \in [M]$, which advocate action a_i (i.e., $cl.a = a_i$):

$$P(s_t, a_i) = \frac{\sum_{cl \in [M]|_{a_i}} cl.p \times cl.F}{\sum_{c \in [M]|_{a_i}} c.F}, \quad \ldots\ldots (11)$$

where $[M]|_{a_i}$ represents the subset of the classifiers of $[M]$ with action a_i, p_k represents the prediction of the classifier cl_k, and F_k represents the fitness of the classifier cl_k. Next, the XCS selects an action to perform. The classifiers in $[M]$ advocate the selected action from the current action set $[A]$. The selected action is performed in the environment, and a scalar reward r_t is returned to the XCS together with a new input configuration.

3.2.2. Reinforcement Component

When the reward r_t is received and the match set $[M]$ with respect to the resulting sensory input is formed, the parameters of the classifiers in $[A]$ are updated in the following order [11]: prediction, prediction error, action set size, and finally, fitness.

The prediction $cl.p$ of each classifier cl in $[A]$ is updated with the learning rate β $(0 < \beta \leq 1)$ and discount rate γ $(0 < \gamma \leq 1)$. If the system solves a supervised classification (single-step) problem or the termination criterion is met in a reinforcement learning (multi-step) problem, the prediction $cl.p$ of each classifier in $[A]$ is updated with the current reward r_t. Otherwise, it updates the classifiers in the previous action set $[A]_{-1}$, which is the action set of previous step with the previous reward r_{t-1}, as follows,

$$P = \begin{cases} r_t & \text{(if the termination criterion is met)} \\ r_{t-1} + \gamma \times \max_a P(s_t, a) & \text{(otherwise)} \end{cases} \quad (12)$$

$$cl.p \leftarrow cl.p + \beta(P - cl.p) \quad \ldots\ldots\ldots (13)$$

Then, the prediction error $cl.\varepsilon$ and the action set size $cl.as$ of each classifier cl are updated as follows:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p| - cl.\varepsilon) \quad \ldots\ldots (14)$$

$$cl.as \leftarrow cl.as + \beta\left(\left[\sum_{c \in [A]} c.num\right] - cl.as\right) \quad . \quad (15)$$

Finally, the classifier fitness is updated in two steps: first, the accuracy $cl.\kappa$ of the classifier in $[A]$ is computed as follows,

$$cl.\kappa = \begin{cases} 1 & \text{if } cl.\varepsilon < \varepsilon_0 \\ \alpha\left(\dfrac{cl.\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{otherwise.} \end{cases} \quad \ldots\ldots (16)$$

The accuracy $cl.\kappa$ implies that a classifier is considered accurate if its prediction error $cl.\varepsilon$ is smaller than the threshold $cl.\varepsilon_0$; a classifier that is accurate has an accuracy $cl.\kappa$ equal to 1. A classifier is considered to be inaccurate if its prediction error $cl.\varepsilon$ is larger than $cl.\varepsilon_0$; the accuracy $cl.\kappa$ of an inaccurate classifier is computed as a potentially descending slope given by $\alpha(cl.\varepsilon/\varepsilon_0)^{-\nu}$. Then, the fitness $cl.F$ of each classifier cl in $[A]$ is updated with a *relative accuracy* $cl.\kappa'$ as follows:

$$cl.\kappa' = \frac{cl.\kappa \times cl.num}{\sum_{c \in [A]} c.\kappa \times c.num} \quad \ldots\ldots\ldots (17)$$

$$cl.F \leftarrow cl.F + \beta(cl.\kappa' - cl.F). \quad \ldots\ldots\ldots \quad (18)$$

3.2.3. Discovery Component

On a regular basis, a genetic algorithm (GA) is applied to the classifiers in $[A]$ or a previous action set $[A]_{-1}$ depending on the parameter θ_{GA}. It selects two classifiers based on the fitness of the classifiers $[A]$, copies them, and performs crossover and mutation on the copies with probability χ and μ respectively. The resulting offspring are inserted into the population and two classifiers are deleted if the number of classifiers in the population $[P]$ is larger than the *population size limit N* to keep the population size constant. This work uses two-point crossover, niche mutation [11], and a tournament selection [12] in all systems.

3.2.4. Subsumption

The *subsumption* operation is applied to the classifiers in $[A]$ after updating the classifier parameters and to the offspring after GA. A classifier can be subsumed by a more general classifier than itself, provided that the more general classifier is accurate and well updated (i.e., $\varepsilon \leq \varepsilon_0$, $\exp > \theta_{sub}$).

3.3. XCS to XCSR

The XCSR is the XCS extension for continuous real value problems.

3.3.1. Classifier Representation

In the XCSR, the classifier condition is changed. The XCSR expresses the conditions with a range of continuous values called the CS expression. The CS expression has two values – a center value (c_i) and a spread value (s_i) – and the range is $[l_i, u_i)$, where l_i is $c_i - s_i$ and u_i is $c_i + s_i$. Then, the classifier matches the inputs if all values in the inputs are within the condition range (i.e., $l_i \leq x_i < u_i$ for all x_i). When covering the classifiers, the conditions for the new classifier are $c_i = x_i,\ s_i = \text{random}(s_0)$. x is an input, and $\text{random}(s_0)$ represents a real random number from 0 to s_0. s_0 is a covering system parameter in the XCSR.

3.3.2. Rule-Discovery Component

In the rule-discovery component, the mutation operator is modified as $c_i \leftarrow c_i + \text{random}(2m) - m,\ s_i \leftarrow s_i + \text{random}(2m) - m$. m is a covering system parameter in the XCSR.

3.3.3. Subsumption of XCSR

The subsumption mechanism of the XCS is modified such that a classifier cl_1 of a bigger interval range $[l_i, u_i)$ of condition can subsume a classifier cl_2 of smaller interval range $[l_i', u_i')$ of condition (i.e., $l_i \leq l_i', u_i' \leq u_j$).

4. Proposed System

4.1. Features

The classification correctness rate of a neural network is high, and it derives the correct output from some unknown input: however, it is difficult for a neural network to derive the correct output for an unknown low frequency outlier-like input. On the other hand, because the LCS is an evolutionary rule-based system, it is easy for the LCS to adapt to low frequency outlier-like inputs by generating new classifiers; however, it is difficult for the LCS to handle high-dimensional input because it is too large to allow the LCS to explore all best state-action rules.

Because a neural network adjusts the weight parameters to minimize the error of all samples, it is difficult for a neural network to learn low frequency outlier-like input. However, because the LCS is a rule-based classifier system, when an unknown input is encountered, the LCS can deal with it correctly by generating a new classifier, which can match the input by a covering operation. This paper proposes a hybrid system of the deep neural network and the XCSR to improve the correctness rate of classification of the deep neural network. Because it is difficult for the LCS to learn from high dimensional input, the deep neural network extracts features of input and compresses the information to reduce the dimension so that XCSR can learn. Because XCSR can handle low frequency outlier-like input, the proposed system can also handle low frequency outlier-like inputs. Therefore, the proposed system can improve the correctness rate of classification because it can explore unknown solutions.

4.2. Architecture

The proposed system is composed of the XCSR and the improved deep neural network – the Deep Classification Autoencoder (DCA). The architecture of the XCSR is shown in **Fig. 4** in Section 3.

4.2.1. Deep Neural Network Component – The Deep Classification Autoencoder (DCA)

The improved deep neural network for the proposed system is composed of the deep neural network for classification and deep autoencoder. Henceforth, we call the improved it the Deep Classification Autoencoder (DCA). **Fig. 5** shows the architecture of the DCA. The DCA has two output layers – the classification layer and the autoencoder layer. The aim of the DCA is to extract the features of the input with high-dimensional compression. A normal autoencoder can compress the input; however, it does not have class labels. The DCA is designed such that the autoencoder learns the features of the input according to the class. In detail, the first half layers of the autoencoder can compress the input to replace similar common parts found by comparing all inputs with the short symbols. Because the autoencoder reproduces the input information, the first half layers of the autoencoder compresses the input without losing the necessary information. However,

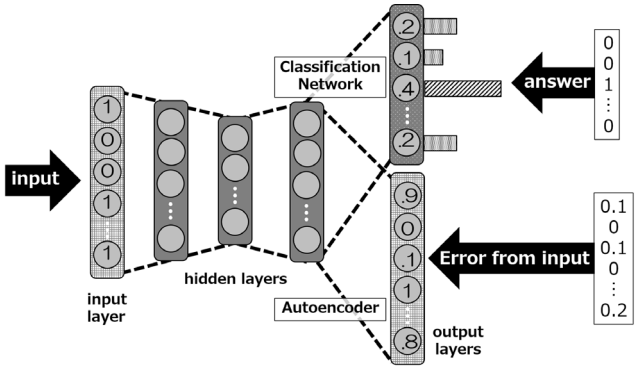


Fig. 5. Architecture of the deep classification autoencoder (DCA).

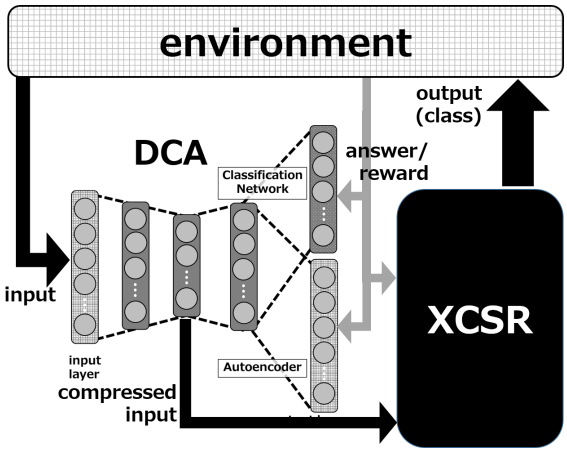


Fig. 6. Architecture of the proposed system.

the compression rate of the autoencoder is small because the autoencoder can compress only the similar parts of the input. On the other hand, the neural network for classification can reduce unrelated information received as input for classification. However, the neural network for classification can be applied for classification problem only. To tackle the problems, by combing the autoencoder and neural network for classification, the first half layers of the autoencoder can compress more information of the input without losing necessary information than the autoencoder based on the features for classification derived from the neural network for classification. This is because it is possible to replace the relationship between the inputs obtained by the neural network for classification with a short symbol. The DCA has higher compression performance than the ordinary autoencoder. Increasing the compression ratio makes it possible to reduce the input to fewer dimensions and to improve the learning speed and accuracy of the XCSR, which is normally not good at learning from high-dimensional input.

The output layers of the DCA are fully connected to the common previous layer, and the activation function of the classification output layer employs the softmax function (Eq. (9)); the activation function of the autoencoder output layer employs the logistic sigmoid function (Eq. (3)).

The layer constitution of the DCA is the same as that of deep autoencoder [13], which has an hourglass type structure except for the output layers. The lengths of the nodes of the classification output layer are the number of classes, and the lengths of the nodes of the autoencoder output layer are the same as that of the input layer.

The outputs of the DCA are the class possibility distribution and the decoded input; however, the proposed system does not use both outputs but uses the compressed input in the hidden layer.

4.2.2. Abstract of the System

The proposed system is composed of the DCA and the XCSR. **Fig. 6** shows the entire system architecture.

The first half layers of the DCA play the role of an encoder, which compresses the environmental input. The input to the XCSR is compressed input generated by the encoder. The XCSR gets the reward from the environment by outputting an action to the environment and updates the rules according to the given reward. The environment also provides answer in order to learn the DCA. The output of the XCSR is the output of the proposed system; note that the output of the DCA is ignored.

4.3. Mechanism

4.3.1. Learning of the DCA

The principle of the learning mechanism is the same as that of the autoencoder and neural network except for the output layers. This is because both the autoencoder and neural network in the DCA share the hidden and input layers but separately have their own output layers. The activate and error functions of the autoencoder layer are used in Eqs. (2) and (4), and the classification layer is used in Eqs. (9) and (10). When the DCA learns, it does not execute the learning of the autoencoder and neural network separately, but regards two output layers as a single output layer and executes the backpropagation as the learning to change the weight of the entire system. Note that both output layers do not interfere with each other because they are independent of each other. In the learning, the output layer of the autoencoder is set as the data in the input layer, whereas the output layer of the neural network is set as the correct value (label) converted into the 1-of-K representation. Note that the learning of the autoencoder and neural network in the DCA is completed by the learning of the entire system. The brief algorithm for understanding the DCA is summarized as follows: (1) one set of the input and output data is given to the DCA; (2) the input data is set in the input layer in the DCA, whereas the same input data is set in the output layer for the autoencoder in the DCA. The output data (converted into the 1-of-K representation) is set in the output layer for NN in the DCA; and the backpropagation used as the learning is executed over the entire system; (4) the process is repeated until all data sets are input to the DCA.

4.3.2. DCA Component

The DCA reduces the dimensions of the input data to a level that the XCSR can learn, and the compressed data are input to the XCSR. When the DCA is composed of L layers with n nodes in the input layer and $m(n > m)$ nodes in the central layer (the $(L+1)/2$-th layer), the first layer after the central layer is the encoder, which compresses n input data to m data, and the central layer to the L th layer is the decoder, which decodes m data to ($n+$ the number of classes) data along with the output distribution possibility of the class. The input data from the environment are input to the input layer of the DCA and calculation is performed from the input layer to the central layer. The outputs of the central layer are the compressed data and input data of the XCSR.

4.3.3. XCSR Component

The XCSR receives input, which is reduced in dimension by the DCA, and learns the classifiers in the lower dimension. The XCSR receives the input from the DCA, outputs an appropriate action to the environment, gets a reward from the environment, and updates the rules using the reward. The output of the XCSR becomes the output of the proposed system. Note that the XCSR learns only in the low-dimensional space; hence, all operations are done in the low-dimensional space.

4.4. Algorithm

Algorithm 1 shows the algorithm of the proposed system. In **Algorithm 1**, L is the number of layers in the DCA.

1. Initialize the DCA, and conducts pre-training using training samples from the environment. (line 1–9)
2. Learn the DCA using training samples from environment until error of output is not down. After that, the DCA stop learning. (line 10–16)
3. Initialize the XCSR, and learn. Input of XCSR uses output of the encoder in the DCA. (line 17, 19–23)
4. When the system meets the termination conditions, finish learning. (line 18, 24)
5. The output of XCSR is the output of proposed system. (line 21)

Algorithm 1 Proposed system.

```
1:  Initialize weights of DCA according to gaussian distribution
2:  for i = 0...L-2 do
3:    AE[i] ← Make Autoencoder with DCA. Layer[i] and DCA. Layer[i+1]
4:    while Average error of output is down do
5:      x ← Environment. GetSituation
6:      y ← AE[i]. Output(x)
7:      AE[i]. Backpropagation(x, y)
8:    end while
9:  end for
10: while Average error of output is down do
11:   x ← Environment. GetSituation
12:   c ← Environment. GetClass
13:   y ← DCA. OutputAutoencoderLayer(x)
14:   d ← DCA. OutputClassificationLayer(x)
15:   DCA. Backpropagation(x, y, c, d)
16: end while
17: Initialize XCSR
18: while Not end of learning do
19:   x ← Environment. GetSituation
20:   σ ← DCA. Encode(x)
21:   act ← XCSR. Input(σ)
22:   ρ ← Environment. Execute(act)
23:   XCSR. Update Parameters(ρ)
24: end while
```

5. Experiment

To confirm that 1) the compression function of the DCA works well, 2) the proposed system performs in a superior manner, even learning from the compressed inputs, and 3) the classification correctness rate of proposed system is better than that of the neural network (DCA) and XCSR, we conducted an experiment on a classification benchmark problem – Connectionist Bench (Sonar, Mines vs. Rocks) [14].

5.1. Connectionist Bench (Sonar, Mines vs. Rocks)

Connectionist Bench (Sonar, Mines vs. Rocks) is a classification benchmark problem which is used in [14]. The problem involves training a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off an approximately cylindrical rock.

There are 60 attributes all of which are continuous real values in range of from 0 to 1. There are two classes (mines and rocks) and 208 instances (111 mine instances and 97 rock instances).

As 208 instances are too few to train neural network, in this experiment, we created a dataset from the original dataset using the following method: (1) copy a data point of the original dataset; (2) add a random value in the range from -0.05 to 0.05 to all attributions; (3) repeat these steps 99 times for each data point. By employing this process, the dataset size increase to 20800.

5.2. Evaluation and Experiment Settings

To compare the respective method (i.e. the proposed system) and the XCSR, we use the correctness rate of classification as evaluation criteria, which is the correct answer rate in 10000 random connectionist bench problems.

The DCA is prior-trained by stacking autoencoder in an appropriate manner, and a suitable amount of learning is carried out, until the error is sufficiently small. Each layer of the DCA is composed of {60-24-10-24-62} nodes. All weights are initialized according to Gaussian distribution,

Table 1. Parameters of XCSR.

Parameter	Explanation	Value
N	maximum number of classifiers in $[P]$	1000
β	learning rate	0.2
θ_{GA}	threshold of GA	25
χ	crossover rate	0.8
μ	mutation rate	0.04
θ_{del}	threshold of classifier deletion	20
δ	parameter of classifier deletion	0.1
θ_{sub}	threshold of subsumption	20
ε_0	allowable error	10
α	parameter of calculating F	0.1
ν	parameter of calculating F	5
p_I, ε_I, F_I	initial value of new classifier	all 0.01
θ_{mna}	minimum number of actions in $[M]$	2
p_{explr}	probability to act randomly	1
S_0	maximum value of s_0	1.0
M	maximum value of m	0.1

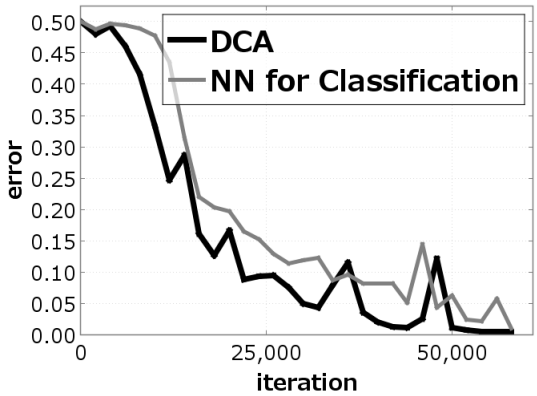


Fig. 7. Classification error of neural networks in each iteration.

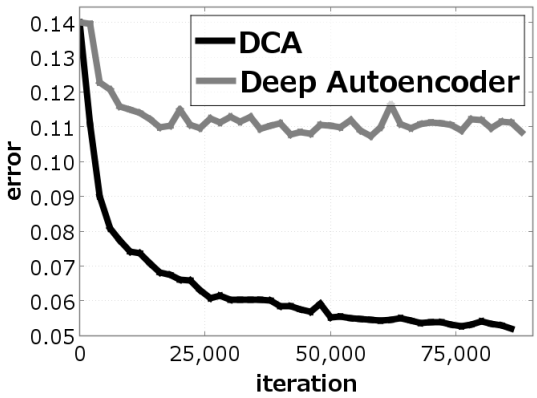


Fig. 8. Reproduction error of autoencoders in each iteration.

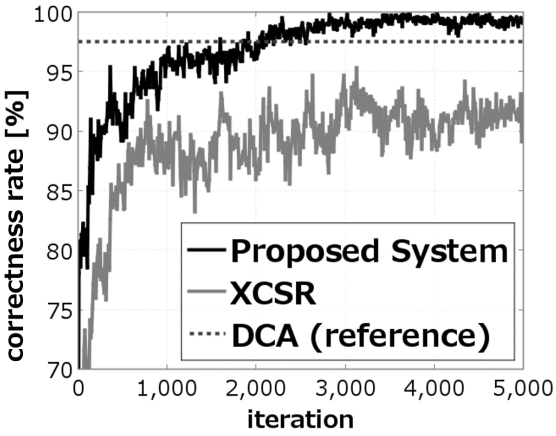


Fig. 9. Classification correctness rate of each system.

and all biases are initialized to 0. The learning rate of the DCA (ε) is set to 0.02.

The normal XCSR is input 60-dimensional continuous real values, and the proposed system compresses 60-dimensional continuous real values to 10 dimensions of continuous real values. The XCSR in the proposed system is input compressed 10-dimensional continuous real values.

The general parameters of XCSR are set as listed in **Table 1**.

The parameters of conventional methods, i.e. deep neural network and deep autoencoder, are set to the same as DCA.

5.3. Results

Figure 7 shows the result of the "classification error" of DCA and the neural network, both of which have the same parameters. In this figure, the vertical and horizontal axes indicate the classification error and iteration, respectively. Note that the small classification error means that the input data is correctly classified with a high probability. **Fig. 7** shows that the classification error of DCA and neural network are approximately the same, which suggests that DCA can compress the input data with remaining necessary features with the same proficiency as that of the neural network.

Next, **Fig. 8** shows the result of the "reproduction error" of DCA and autoencoder, both of which have the same parameters. In this figure, the vertical and horizontal axes indicate the reproduction error and iteration, respectively. Note that a system which has a smaller reproduction error can compress the input data at a higher rate than a system which has a larger reproduction error (i.e., it is possible for the former system to compress the input data into smaller dimensions with the same error as that of the latter system). **Fig. 8** shows that the reproduction error of DCA is smaller than that of autoencoder, which suggests that DCA can compresses the input data at a higher compression rate in comparison with autoencoder. From **Figs. 7** and **8**, DCA can compress the input data with remaining necessary features proficiently as the of neural network and its compression rate is higher than that of autoencoders.

Figure 9 shows each individual correctness rate of classification which is evaluated at each XCSR iteration. The vertical and horizontal axes indicate the correctness rate of classification in 10000 random tests of the respective

Table 2. Converged correctness rate of classification.

	DCA	XCSR	Proposed system
Correctness rate	97.52%	95.30%	99.90%

system (the normal XCSR, the proposed system) and iterations of XCSR learning, respectively. The gray dotted horizontal line denotes the correctness rate of classification of the DCA, which is 97.52%. It is fixed because it is not affected by the specific iteration of XCSR learning.

Table 2 lists the converged maximum correctness rate of classification for various systems (the DCA, the normal XCSR, and the proposed system).

As observed from **Fig. 9** and **Table 2**, the proposed system is always more accurate than the normal XCSR, and the correctness rate of the proposed system became higher than the correctness rate of the DCA from 26,000 iterations.

5.4. Discussion

Figures 7 and **8** show that the DCA can compress input with remaining necessary features like autoencoder, and its compression rate is higher than that of the normal deep autoencoder without losing the accuracy of classification.

Figure 9 shows that the XCSR in the proposed system can learn correctly, even learning from the compressed inputs.

In **Table 2**, it can be observed that the correctness rate of the normal XCSR converges to 95.30%. The reason it does not converge to 100% is the shortage of N (maximum number of classifiers in $[P]$). As the proposed system compresses 60 dimensions to 10 dimensions, the $N = 1000$ is sufficient for the proposed system. On the other hand, the normal XCSR is input a 60-dimensional input, and it needs many classifiers to adapt the environment, however the N is limited the number of classifiers makes this result. Hence the correctness rate of classification of the proposed system is always higher than the normal XCSR's one, as shown in **Fig. 9**.

The reason that the correctness rate of classification of the proposed system is higher than that of the DCA is that XCSR can deal with low frequency outlier-like input by generating fit classifiers or evolves classifiers. The neural network can output correct answers when the features of input are similar to the learned features. However, when the system receives low frequency outlier-like input, sometimes neural network gives incorrect results. For example, if the DCA (or the neural network) learns features such as "if the spectrum of the high frequency component is strong, the output is *mine*," it cannot output the correct answer when exceptions are input such as "however, the spectrum of the high frequency component is strong, the *rock* is correct." The XCSR can learn it correctly by generating it as a specialized classifier. In this result, we confirmed that inputs for which the DCA does not output correct answer could be adequately dealt with by the proposed system in covering operations in the XCSR component. The covering operation in the XCSR is an operation

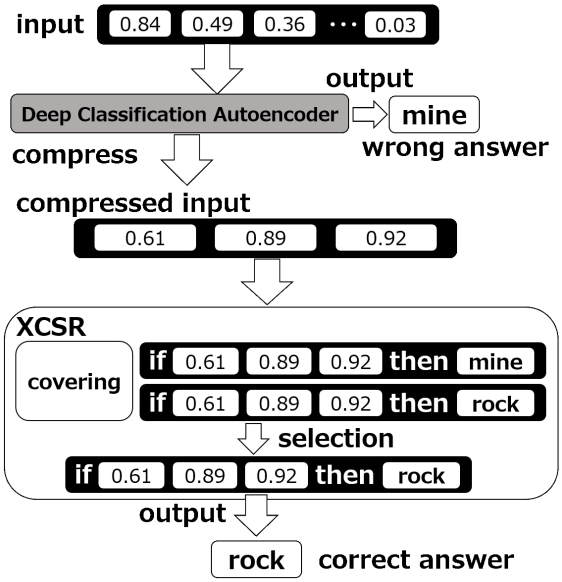


Fig. 10. An image of modifying answer by covering mechanism when DCA outputs wrong answer.

that generates classifiers to match inputs. The proposed system outputs correct output by the covering operation in the proposed system even if the DCA outputs wrong output. **Fig. 10** shows an example image in which the wrong outputs by DCA are modified to the correct answer by the covering operation. In this figure, the following operations are conducted; 1) 60-dimensional continuous real values from the environment (*0.84, 0.48, 0.36, ..., 0.03*) are input to the DCA; 2) the output classification layer of the DCA outputs estimates class (*"mine"*) from the input, and at the same time outputs compressed continuous real values by the central layer of the DCA (*0.61, 0.89, 0.92*); 3) the covering operation of XCSR generates classifiers which cover compressed input ({*if "0.61, 0.89, 0.92" then "mine"*} and {*if "0.61, 0.89, 0.92" then "rock"*}); 4) the reinforcement component of XCSR updates fitness of classifiers; 5) the performance component selects the best action ({*if "0.61, 0.89, 0.92" then "rock"*}), and outputs the correct answer (*"rock"*).

The reason that the correctness rate of classification did not became 100% in the XCSR of the proposed system is thought to be related to the similarity of features when the different input is compressed by the DCA. However, because they are not all the same features, the correctness rate seems to be improved by updating parameters of classifiers of the XCSR by repeating learning.

In addition, we conducted the same experiment in the conventional deep neural network for classification and the XCSR with conventional deep neural network for classification to compare the proposed system against them. **Table 3** shows converged correctness rate of classification on the conventional deep neural network for classification and XCSR with conventional deep neural network for classification in the same experiment conditions. From this table, even this is not as much as the proposed system. However the correctness rate of classification of the

Table 3. Converged correctness rate of classification in conventional neural network for classification and its hybrid with XCSR.

	NN	NN with XCSR
Correctness rate	96.81%	97.98%

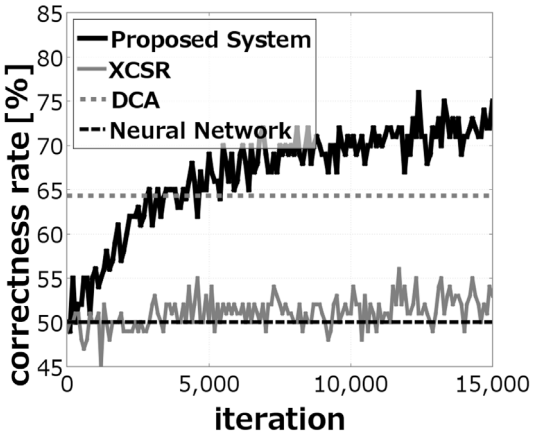


Fig. 11. Correctness rate of 20-multiplexer problem.

conventional deep neural network for classification is improved by the XCSR, and the effectiveness of combining neural network and the LCS was shown. It is said that deciding parameters of neural network (including layer size and structure) is difficult. However, the XCSR can cover this problem. We can also observe that the DCA can compress input more effectively than the conventional neural network and it makes the DCA more accurate.

Finally, in order to investigate the generalization capability of the proposed system, we additionally conducted the experiment of 20-multiplexer problem [15] as it differs from the problem of the Connectionist Bench problem. The 20-multiplexer problem is the binary classifier problem as one of the major LCS benchmark problems. The 20-multiplexer problem has 20 binary attributes. The first 4 bits are called address bits, and the other 16 bits are called reference bits. The answer of input is represented in the reference bit which is indicated by the address bits. Unlike the Connectionist Bench problem where the significant attributes (which give a big influence on the output) are fixed to all input data, such attributes vary according to the input data in the 20-multiplexer problem, which make it a difficult to solve using a neural network. The experiment settings follow Section 5.2 except for $N = 30000$. We compared the correctness rate of the proposed system with DCA, XCSR, and neural network. **Fig. 11** shows the classification result, where the vertical and horizontal axis indicate the correctness rate of the classification and the learning iteration, respectively. From this figure, the correctness rate of classification is high in the case of normal neural network, XCSR, DCA and the proposed system. This result indicates that the correctness rate of classification by the proposed system is higher than that of the other methods for the 20-multiplexer problem, which is the same as the tendency that is shown in the case of the Connectionist Bench problem. These implications suggest that the proposed system has a generalization capability which is higher than the other comparable methods.

As can be observed in this figure, the correctness rate of classification is high in the case of neural network for classification, normal XCSR, DCA, and the proposed system. The tendency which the correctness rate of the proposed system is higher than that of DCA, XCSR, and neural network for classification is the same as that shown in the case of the Connectionist Bench problem. These experimental results suggest that the proposed system is applicable to various data sets regardless of the continuity or discontinuity of values. However, it is difficult for the proposed system to learn from datasets that cannot be compressed by feature extraction, which correlates to all input values like parity problems.

Based on these reasons, the following are the implications 1) the reproduction error of DCA is smaller than the normal deep autoencoder, 2) the proposed system performs well, even learning from the compressed inputs, 3) the classification correctness rate of proposed system is better than that of the DCA and the XCSR, by dimension reduction and covering operations of the XCSR, and 4) the N (maximum size of classifier) of the proposed system needed is lower than that of the normal XCSR.

6. Conclusion

This paper first proposed the Deep Classification Autoencoder (DCA), which is a dimension compressor with necessary features by combining the deep neural network for classification and autoencoder. Then, it proposed the XCSR with the DCA, which is one of LCSs that can deal with continuous real numbers to improve the correctness rate of classification by adapting unknown low frequency outlier-like input.

As a consequence of the proposed system, 1) environmental inputs are encoded by the first half layers (the encoder) of the DCA; 2) the encoded (compressed) data become the input of the XCSR; 3) XCSR outputs the action(class) to the environment; and 4) XCSR receives the reward from the environment and learns classifiers.

To verify the effectiveness of the proposed system, we conducted an experiment on the Connectionist Bench problem that compares the results of XCSR which learns from uncompressed data (60 continuous real values) with those of the proposed system which learns from the data compressed from 60- to 10-dimensional data (continuous real values). Through intensive empirical experiments, the following implications have been revealed; 1) the reproduction error of DCA is smaller than that of the normal deep autoencoder; 2) the proposed system performs better than the normal XCSR, even learning from the compressed input data; and 3) the proposed system performs better than the conventional neural network by deriving

correct output in the case of low frequency outlier-like input. These results show the proposed system can solve some problems that neural network can not solve.

Future work can focused on that to activating interpretability of LCS in the proposed system, so that the DCA can decode classifiers to generalized human readable if-then rules.

References:

[1] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, Vol.18, No.7, pp. 5271-1554, 2006.

[2] H. John, "Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," Machine learning, an artificial intelligence approach, Vol.2, pp. 593-623, 1986.

[3] S. W. Wilson, "Get real! XCS with continuous-valued inputs," Learning Classifier Systems, pp. 209-219, Springer, 2000.

[4] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," Evolutionary computation, Vol.10, No.2, pp. 99-127, 2002.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," Nature, Vol.518, No.7540, pp. 529-533, 2015.

[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine learning, Vol.3, No.1, pp. 9-44, 1988.

[7] G.W. Cottrell and P. Munro, "Principal components analysis of images via back propagation," Visual Communications and Image Processing '88: 3rd in a Series, pp. 1070-1077, International Society for Optics and Photonics, 1988.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, Vol.5, No.4, pp. 115-133, 1943.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Cognitive Modeling, Vol.5, No.3, 1988.

[10] S.W.Wilson, "Classifier Fitness Based on Accuracy," Evolutionary Computation, Vol.3, No.2, pp. 149-175, June 1995.

[11] M. V. Butz and S. W. Wilson, "An Algorithmic Description of XCS," Soft Computing, Vol.6, No.3.4, pp. 144-153, 2002.

[12] M. V. Butz, D. E. Goldberg, and K. Tharakunnel, "Analysis and Improvement of Fitness Exploitation in XCS: Bounding Models, Tournament Selection, and Bilateral Accuracy," Evolutionary Computation, Vol.11, No.3, pp. 239-277, 2003.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, Vol.313, No.5786, pp. 504-507, 2006.

[14] R. P. Gorman and T. J. Sejnowski, "Analysis of hidden units in a layered network trained to classify sonar targets," Neural Networks, Vol.1, No.75, 1988.

[15] S. W. Wilson, "Classifier fitness based on accuracy," Evolutionary computation, Vol.3, No.2, pp. 149-175, 1995.

Name:
Kazuma Matsumoto

Affiliation:
The University of Electro-Communications

Address:
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan

Brief Biographical History:
2016- Graduated the University of Electro-Communications Bachelor's 4th grade
2016- Enrolled the University of Electro-Communications Master's Course

Main Works:
• Matsumoto, K. et al., "Learning classifier system with deep autoencoder," Evolutionary Computation (CEC), 2016 IEEE Congress on. IEEE, 2016.

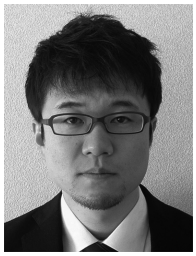
Name:
Takato Tatsumi

Affiliation:
The University of Electro-Communications
Research Fellow of Japan Society for the Promotion of Science

Address:
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan

Brief Biographical History:
2015-2017 Master Course Student, The University of Electro-Communications
2017- Doctoral Course Student, The University of Electro-Communications
2017- Research Fellow, Japan Society for the Promotion of Science

Main Works:
• "A Learning Classifier System that Adapts Accuracy Criterion," Trans. of the Japanese Society for Evolutionary Computation, Vol.6, No.2, pp. 90-103, 2015.



Name:
Hiroyuki Sato

Affiliation:
The University of Electro-Communications

Address:
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan

Brief Biographical History:
2009- Assistant Professor, Faculty of Electro-Communications, The University of Electro-Communications
2010- Assistant Professor, Graduate School of Informatics and Engineering, The University of Electro-Communications
2016- Associate Professor, Graduate School of Informatics and Engineering, The University of Electro-Communications

Main Works:
• H. Sato, "Chain-Reaction Solution Update in MOEA/D and Its Effects on Multi and Many-Objective Optimization," Soft Computing, Springer, Vol.20, Issue 10, pp. 3803-3820, 2016.

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE)

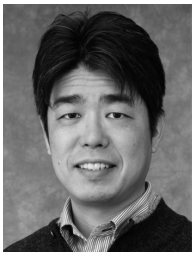
Name:
Tim Kovacs

Affiliation:
Honorary Senior Lecturer, Department of Computer Science, University of Bristol

Address:
MVB, Woodland Rd., Bristol, BS8 1UB, United Kingdom

Brief Biographical History:
2001- Joined the University of Bristol
2006- Visiting Scientist, University of New South Wales
2015- Visiting Scientist at the University of Electro-Communications

Main Works:
• Kovacs, T., "Strength or Accuracy: Credit Assignment in Learning Classifier Systems," Springer-Verlag, ISBN1-85233-770-2, 2004.



Name:
Keiki Takadama

Affiliation:
The University of Electro-Communications

Address:
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan

Brief Biographical History:
1995- M.E. degree at Kyoto University
1998- Ph.D. at the University of Tokyo
1998-2002 Visiting Researcher, Advanced Telecommunications Research Institute (ATR) International
2002-2006 Lecturer, Tokyo Institute of Technology
2006-2011 Associate Professor, The University of Electro-Communications
2011- Professor, The University of Electro-Communications

Main Works:
• Evolutionary computation, reinforcement learning, multiagent system, healthcare system

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE)
• Association for Computing Machinery (ACM)
• AI- and informatics-related academic societies in Japan