XCSR Based on Compressed Input by Deep Neural Network for High Dimensional Data

Kazuma Matsumoto
The University of Electro-Communications
Tokyo, Japan
kazuma@cas.lab.uec.ac.jp

Ryo Takano
The University of Electro-Communications
Tokyo, Japan
takano@cas.lab.uec.ac.jp

Takato Tatsumi
The University of Electro-Communications
Tokyo, Japan
tatsumi@uec.ac.jp

Hiroyuki Sato
The University of Electro-Communications
Tokyo, Japan
h.sato@uec.ac.jp

Tim Kovacs
University of Bristol
Bristol, United Kingdom
tim.kovacs@bristol.ac.uk

Keiki Takadama
The University of Electro-Communications
Tokyo, Japan
keiki@hc.uec.ac.jp

ABSTRACT

This paper proposes the novel Learning Classifier System (LCS) which can solve high-dimensional problems, and obtain human-readable knowledge by integrating deep neural networks as a compressor. In the proposed system named DCAXCSR, deep neural network called Deep Classification Autoencoder (DCA) compresses (encodes) input to lower dimension information which LCS can deal with, and decompresses (decodes) output of LCS to the original dimension information. DCA is hybrid network of classification network and autoencoder towards increasing compression rate. If the learning is insufficient due to lost information by compression, by using decoded information as an initial value for narrowing down state space, LCS can solve high dimensional problems directly. As LCS of the proposed system, we employs XCSR which is LCS for real value in this paper since DCA compresses input to real values. In order to investigate the effectiveness of the proposed system, this paper conducts experiments on the benchmark classification problem of MNIST database and Multiplexer problems. The result of the experiments shows that the proposed system can solve high-dimensional problems which conventional XCSR cannot solve, and can obtain human-readable knowledge.

CCS CONCEPTS

• **Computing methodologies → Rule learning**;

KEYWORDS

LCS, XCS, XCSR, Neural Network, Deep Learning

ACM Reference Format:
Kazuma Matsumoto, Ryo Takano, Takato Tatsumi, Hiroyuki Sato, Tim Kovacs, and Keiki Takadama. 2018. XCSR Based on Compressed Input by Deep Neural Network for High Dimensional Data. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208281

1 INTRODUCTION

Knowledge discovery techniques are attracted attention these days because of its growth of information technology. Learning Classifier System (LCS) [8] which is one of knowledge acquiring system is evolutionary rule-based machine learning using reinforcement learning. LCS gains generalized if-then rules called classifiers through generating, deleting, updating them. Since generalized classifier can be matched to more than one input, thus it represent a common pattern in a problem, LCS is one of useful knowledge discovery systems, and can provide human-readable knowledge with its reliable generalization capability.

However, it is hard for LCS to handle high-dimensional problems (dataset) because LCS need to generate all rules for matching any state space for learning. Correspondingly, some LCS works study how LCS can handle the high-dimensional dataset effectively. For instance, Iqbal proposed LCS extension which reuse knowledge learned in low-dimensional problem for high-dimensional problems [7]. Abedini introduced the guided rule that evolves rules independently representing elements of input [1] in order to improve on a learning efficiency. However, due to the complexity of the systems, these approaches are limited in applicable problems.

This paper proposes the hybrid system of the neural network and LCS for handling common high-dimensional problems and focuses on compression of high-dimensional input to lower. In details, we use deep neural network and XCSR[16] which is LCS extension for continuous real number. The deep neural network extracts features of high-dimensional input, compresses it, and they are input its features to XCSR to learn. After learning classifiers in compressed low-dimension, compressed classifiers are decompressed (decoded) to original dimension by the deep neural network in order to acquire human-readable rules.

In order to investigate the proposed system, this paper conducts experiments on the benchmark classification problem of MNIST dataset [9] and Multiplexer problems [14]. The problem of MNIST dataset is to classify hand written digits, and it has 784 dimensions.

The Multiplexer problem is difficult problem which input cannot be compressed uniformly.

This paper is organized as follows. Section 2 and 3 describe mechanism of XCSR and the deep neural network respectively. In section 4, we explain the detailed mechanism of proposed system, the experiment is conducted in section 5 on a benchmark classification problem. Finally, we conclude this paper in section 6.

2 XCSR

XCSR is the extension of the XCS [14] (i.e., one of major LCS for binary input) with a continuous real-valued coding for classifiers. XCS is a reinforcement learning [13] method in which generalization is obtained through the evolution of a population $[P]$ which stored classifiers.

2.1 Classifier

The classier is the if-then rule which has several parameters. In XCS, classifiers consist of a condition C, an action A, and four 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; and finally (iv) the numerosity n, which indicates how many classifiers with the same condition and the same action are merged to this classifier. XCSR expresses conditions with the range of continuous values called CS (Center / Spread) expression. CS expression has 2 values – the center value (c_i) and the 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 input if all values in inputs are in the condition range (i.e., $l_i \leq x_i < u_i$ for all x_i). When there are no classifiers to match with input, XCSR generates classifiers which match to the input. This operation is called "covering". When conducting covering operation, condition of generated classifier is set to $c_i = x_i,\ s_i = \text{random}(s_0)$, where x is a input, and random(s_0) means real random number from 0 to s_0. s_0 is a covering parameter of XCSR.

2.2 Mechanism

XCSR is composed of performance, reinforcement, discovery components and subsumption operation. Although XCSR corresponds to both multi-step and single step problems, we explain here as assuming single step problem. XCSR learns by updating classifiers through repeating the sequence of performance, reinforcement, discovery components.

2.2.1 *Performance Component.*

At each time step, XCSR 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, covering operation takes place and creates a set of classifiers that match and cover all the missing actions. For each possible action A_i in $[M]$, XCSR computes the system prediction $P(A_i)$ which estimates the payoff that XCSR expects if action A_i is performed at that time. The system prediction is computed as the fitness weighted average of the predictions of classifiers in $[M]$, $cl \in [M]$, which advocate action A_i (i.e., $cl.A = A_i$):

$$P(A_i) = \frac{\sum_{cl \in [M] | cl.A = A_i} cl.p \cdot cl.f}{\sum_{cl \in [M] | cl.A = A_i} cl.f} \tag{1}$$

where $cl \in [M] | cl.A = A_i$ represents the subset of classifiers (cl) of $[M]$ with action A_i. Next, the action which has maximum $P(A_i)$ is the selected by XCSR to perform. The classifiers in $[M]$ which advocate the selected action form the current action set $[A]$. The selected action is performed in the environment, and a scalar reward r is returned to XCSR.

2.2.2 *Reinforcement Component.*

When the reward r is received, the parameters of the classifiers in $[A]$ are updated.

The p, ε, as (i.e., which estimates the average size of the action sets this classifier has belonged to) of each classifier cl in $[A]$ is updated according to reinforcement learning with learning rate β ($0 < \beta \leq 1$) as follows.

$$cl.p \leftarrow cl.p + \beta(r - cl.p) \tag{2}$$

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|r - cl.p| - cl.\varepsilon) \tag{3}$$

$$cl.as \leftarrow cl.as + \beta(\sum_{c \in [A]} c.n - cl.as) \tag{4}$$

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

$$cl.\kappa = \begin{cases} 1 & \text{if } cl.\varepsilon < \varepsilon_0 \\ \alpha(cl.\varepsilon/\varepsilon_0)^{-\nu} & \text{otherwise} \end{cases} \tag{5}$$

where $\alpha, \nu, \varepsilon_0$ are parameters of XCSR which constitutes the target error level. The accuracy $cl.\kappa$ means that a classifier is considered to be 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. Then the fitness $cl.f$ of each classifier cl in $[A]$ is updated as follows.

$$cl.f \leftarrow cl.f + \beta(cl.\kappa \cdot \frac{cl.n}{\sum_{c \in [A]}(c.\kappa \cdot c.n)} - cl.f) \tag{6}$$

2.2.3 *Discovery Component.*

On a regular basis depending on the parameter θ_{ga}, a genetic algorithm (GA) [4] is applied to classifiers in $[A]$. It selects two classifiers based on the fitness of classifiers in $[A]$, copies them, and performs crossover and mutation on the copies with probability χ and μ respectively. Mutation is conducted as $c_i \leftarrow c_i + \text{random}(2m) - m$, $s_i \leftarrow s_i + \text{random}(2m) - m$, where m is a mutation parameter in XCSR. The resulting offspring are inserted into the population and classifiers are deleted if the number of classifiers in the population $[P]$ is larger than a population size limit N to keep the population size constant. This work uses two-point crossover, and a roulette wheel selection [2] in all systems.

2.2.4 *Subsumption.*

The subsumption operation is applied to the classifiers in $[A]$ after updating the classifiers' parameters and to offspring after GA. A classifier can be subsumed by a more general classifier than it, provided that the more general classifier is accurate and well-updated (i.e., $\varepsilon \leq \varepsilon_0$, $exp > \theta_{sub}$) where exp is number of times which the classifier belongs in $[A]$. In details, classifier cl_1 of 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., $l_i \leq l'_i, u'_i \leq u_j$). The n of the subsuming classifier is added with n of the subsumed classifier, and the subsumed classifier is deleted from the $[P]$.

3 NEURAL NETWORK

In this section, we explain neural network, and an autoencoder [3] which is one of architecture of the neural network.

3.1 Deep Learning

Neural network is a classifier which is modeled on brain of human [11]. Neural network is composed of nodes, layers, and its connections.

- input layer is the first set of node sensing an environmental input $\vec{x} = \{x_1, x_2, \cdots, x_n\}$ where n is a length of input.
- hidden layer is a set of node that converts the sensed input to the output. Each hidden layer is connected with the neighbor layers. (i.e., the input, other hidden, or output layers.) Each hidden layer represents an internal expression of input inherited from the expression of previous layer. Typical neural networks have equal to or more than one hidden layer to apply complex problems.
- output layer is the layer located at the end of an entire network, which outputs class, or values.

Each node at a layer is connected with other node on next layer with a weight value w. The input value of node u is sum of product of output of each previous layer's node z and weight w. The output of node z is an output of activation function $f(u)$ when input the u. Appropriate activation functions are depend on the layer and problems. Then, neural network aims at learning the values of parameters w which are initially set to a random value under the Gaussian distribution. Note that, for the nodes connecting with the input layer, u_i can be the corresponding input's element x_i, and output of output layer's node becomes output of neural network.

Neural network which has more than 2 hidden layers is called deep neural network, and the machine learning of deep neural network is called deep learning [5]. To use (deep) neural networks for classification, softmax function of k th layer (equation 7) is employed as an activation function of the output layer.

$$f(u_{i,k}) = \frac{e^{u_{i,k}}}{\sum_{l=1}^{N} e^{u_{i,l}}} \tag{7}$$

The number of input layer's nodes is set to input dimension, and the number of output layer's nodes is the number of classes. The node of output layer outputs possibility of belong to each class. Sigmoid function shown as equation 8 is often used for activation function of hidden layer.

$$f(u_{i,k}) = \frac{1}{1 + e^{-u_{i,k}}} \tag{8}$$

The output error is calculated with equation 9, where $\vec{t}$ indicates the answer of class expressed with one-hot expression for input $\vec{x}$, $\vec{y}$ indicates the output of the network, K is the number of the classes.

$$E(\vec{x}) = -\sum_{k=1}^{K} t_k \log y_k \tag{9}$$

this error function is called "log-loss" which constitutes a standard loss function in the context of neural network.

Neural network learns to minimize the output error by updating weights. Figure 1 shows an example of deep neural network for classification with handwriting number recognition problem. The

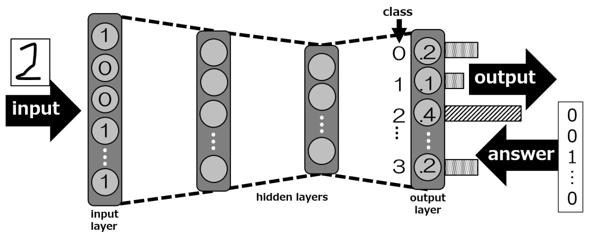

Figure 1: An Example of the Deep Neural Network for Classification (Handwriting Number Recognition)

layer on the left which is input image written the "2" is an input layer, the input propagates through hidden layers and finally, output layer derives the class. In this figure, since the max output value is 2nd units from index of 0, the input image is classified to class "2" by deep neural network. Deep neural network learns to minimize the difference of output of output layer and answer which is converted to one-hot expression. Stochastic gradient descent or its improved method is used for updating weights, in practice, it is done by back propagating the output error from the output layer towards the input layer (backpropagetion [12]).

3.2 Autoencoder

Autoencoder is an algorithm of dimensional reduction using a neural network. The idea of autoencoder is to reproduce the input through by the output, then the learned hidden layers can represent a compressed input. The autoencoder learns the weight values of each node to minimize an error or a difference between the input and the output shown in equation 10.

$$E = \frac{1}{2} \sum_{n=1}^{N} ||\vec{x}_n - \vec{y}(\vec{x}_n)||^2 \tag{10}$$

Sigmoid function (equation 8) is used for hidden layers, and identify function ($f(x) = x$) is used for output layer as activation functions. Figure 2 shows an overview of autoencoder. Specifically, to reproduce the input with the output layer, the number of nodes of the

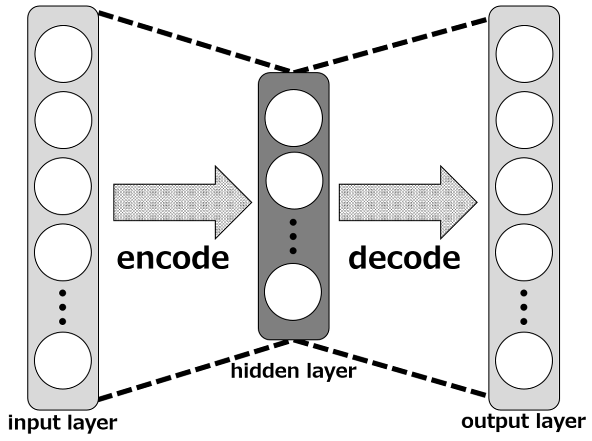

Figure 2: Overview of Autoencoder

output layer is set to the same of the input length. The first half layer of autoencoder encodes input, and last half layer decodes it. With the smaller number of node at the hidden layer than the input length, the hidden layer encodes a sensed input to a low-dimensional input. By inputting encoded (compressed) data to the central layer of autoencoder, decoded data can obtained.

4 PROPOSED SYSTEM: DCAXCSR

The proposed system (DCAXCSR) is a hybrid system of deep neural network called DCA and XCSRs. The aim of DCAXCSR is tackling the problem that LCS cannot deal with high-dimensional input by using compression function of neural network, and gain human-readable rules from high-dimensional problem.

4.1 Deep Neural Network Component — The Deep Classification Autoencoder (DCA)

Deep Classification Autoencoder (DCA) is composed by the deep neural network of classification and deep autoencoder. Figure 3 shows an architecture of the DCA. The DCA has 2 output layers – the classification layer and the autoencoder layer. The aim of the DCA is to extract features of input with high-dimensional compression. Normal autoencoder is able to compress input, however it is not labeled to class. The DCA is designed that autoencoder learns features of input with hint of class. If using only classification layer, the input is too compressed to get features of input. The DCA can compress input with remaining necessary features, and its compression rate is higher than (deep) autoencoders [10]. The output layers of the DCA is fully connected to previous layer, and activation function of the classification output layer employs the softmax function (equation 7), and activation function of autoencoder output layer employs the logistic sigmoid function (equation 8). The layer constitution of the DCA is the same to deep autoencoder [6] which has an hourglass type except output layers. The number of nodes of the classification output layer is the number of classes, and the number of nodes of the autoencoder output layer is the same as input layer. Output of the DCA is the class possibility distribution in the classification output layer and the decoded data in the autoencoder output layer. When learning the DCA, we regard 2 output layers as a single output layer and backpropagates in the same time. The compressed features of input are obtained in the center of hidden layer.

DCA is to reduce dimension of input data which the XCSR can learn, and compressed data are input to XCSR for compressed input. When the DCA is composed of L layers, n nodes in input layer, $m(n > m)$ nodes in central layer (the $(L+1)/2$ th layer), the 1st layer to the central layer are the encoder which compresses n input data to m data and the central layer to the L th layer are decoder which decodes m data to ($n+$ the number of classes) data with output distribution possibility of class. Input data from the environment are input to the input layer of the DCA and calculates from the input layer to the central layer. Output of the central layer is compressed data and it becomes input data of XCSR for compressed input. Decoder decodes condition C of classifiers in population of XCSR for compressed input. To decode, input encoded condition C of classifier to output of $(L+1)/2$ th layer in DCA, decoded data are output to autoencoder output layer.

4.2 Mechanism

4.2.1 Overall System. Figure 4 shows whole system architecture. There are two XCSR, i.e., "XCSR for compressed input" and "XCSR learning from high-dimensional input". The first half layers of the DCA is a role of encoder which compress environmental (high-dimensional) input. XCSR for compressed input is input compressed input by the encoder. XCSR for compressed input gets reward from the environment by outputting action to the environment and updates rules according to be given reward. When the learning of XCSR for compressed input is finished, picks the all learned classifiers in the population, and decodes them by the last half layers of the DCA with autoencoder output layer in order to get human-readable rules.

When XCSR can learn classifiers with high accuracy, system can get human-readable rules by decoding them, however when the learning of classifiers does not go well due to the loss of information by dimensional compression, they become imperfect rules after decoding. However, even imperfect rules can be used to narrow down state space and they can make that learning from high-dimensional possible. The decoded imperfect rules are set to initial population of XCSR learning from high-dimensional input. By learning the

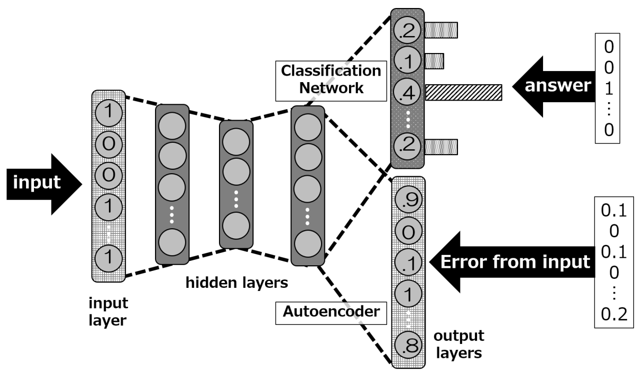

Figure 3: An Architecture of DCA

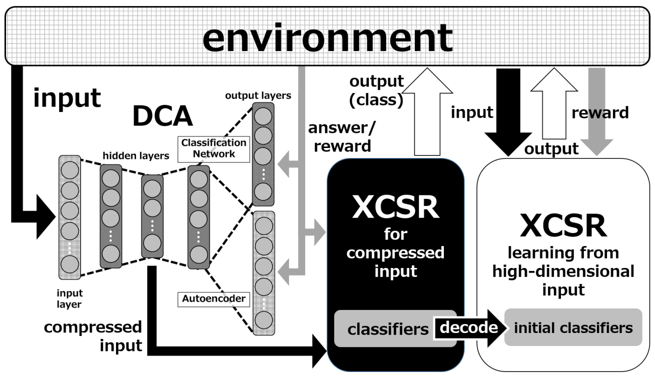

Figure 4: An Architecture of DCAXCSR

initialized system (XCSR learning from high-dimensional input) directly with high-dimensional-input, imperfect rules can be restored and perfect rules can be acquired. This makes the system learnable from high-dimensional input.

4.2.2 XCSR for compressed input. XCSR for compressed input receives input which are reduced dimension by the DCA, and learns classifiers in low dimension. The XCSR for compressed input is input from the DCA, output an appropriate action to the environment, gets reward from the environment and updates classifiers using reward.

If unknown data that DCA has not learned is generated within XCSR, there arises a problem that it cannot be correctly decoded. In order to deal with this problem, XCSR for compressed input has made the following improvements to native XCSR; 1) when running GA, first, encoded values are decoded, crossover and mutation are conducted on the decoded values, and finally, encode the values after GA is completed; 2) when random values is assigned to s values in covering operation, the random values are generated in the high dimension, and encoded values are used. When decoding condition of classifiers, input each values of c and s of classifiers in population of XCSR to the decoder of DCA.

4.2.3 XCSR learning from high-dimensional input. When decoded rules are imperfect, they are set to initial population of XCSR learning from high-dimensional input. Imperfect rules mean classifier that condition is almost correct, however only partly wrong, for example in 6-multiplexer problem, "if #0##1# then 1" is a imperfect rule of "if 10##1# then 1" (For the meaning of symbol '#' and 6-multiplexer problem, please refer to the later sections). XCSR learning from high-dimensional input is same as native XCSR except that the initial population is set to decoded rules. When setting to initial population, XCSR learning from high-dimensional input take over condition-action if-then rule and parameter prediction p of classifiers, and the other parameter of classifier is discarded. XCSR learning from high-dimensional input learn from high-dimensional input directly, and when learning is over, we can get human-readable rules from population.

4.3 Algorithm

Figure 5 shows the algorithm flow of the DCAXCSR. The following explanation numbers are corresponding to the numbers in the figure.

(1) Initialize the DCA, and learns to generate networks by using input from the environment.
(2) Compress the high-dimensional input to low-dimensional by encoder of DCA.
(3) Learn XCSR for compressed input with compressed input. Repeat (2)-(3) until convergence.
(4) Decode the compressed classifiers in the population of XCSR for compressed input by the decoder of DCA. If the decoded classifiers are perfect classifiers, the gained classifiers are human-readable rules.
(5) If the decoded classifiers are imperfect classifiers, set them to the initial population of XCSR learning from high-dimensional input.
(6) Learn XCSR learning from high-dimensional input with high-dimensional input. Repeat learning until convergence. The imperfect rules are modified, and human-readable rules are in the population.

5 EXPERIMENT

To confirm that 1) DCAXCSR performs better classification accuracy than XCSR with high-dimensional benchmark problems and 2) acquired rules are well generalized, human-readable and accurate; we conducted experiments on two classification benchmark problems. We chose the MNIST database [9] for the problem of image input which has high dimension which can be compressed to lower dimension with neural network and 11-multiplexer problem [15] for the problem which is impossible for neural network to compress input.

5.1 Problems

5.1.1 The MNIST Database. The MNIST database of handwritten digits (MNIST stands for Modified National Institute of Standards and Technology) is a dataset of images of handwritten digits like Figure 6. (We call it "MNIST" from here.) Each image has 784 pixels (28×28) which each pixel has integer value from 0 to 255. These images are converted to values of linear 786 dimensions and are granted the label of numbers.

In our experiment, we used data labeled "3" and "8" , and used binarized images like Figure 7. There are 11982 samples of images to train. The reason we used only "3" and "8" is that since they have the same shape of "3", it is easy for human to verify whether DCAXCSR can gain human-readable rules because shared feature is well-known.

5.1.2 Multiplexer Problem. Multiplexer problem [14] (MUX) is a benchmark classification problem. In this problem, the system outputs 1-bit in response to the input of l-bit. l-bit input is composed of m-bit address bits and 2^m-bit reference bits ($l = m + 2^m : m \in \mathbb{N}$). The multiplexer problem with l-bit input is called l-multiplexer problem. The value of the reference bits indicated by the address bits is the answer output. If address bits indicate a in decimal number, the a th reference bit indicates an answer. Fig. 8 shows an example of the 6-multiplexer problem. When the length of input bits is 6 ($l = 6$) and length of address bit is 2 ($m = 2$), address bits of "101110" are "10"=2, and reference bits are "1110", then the answer of "101110" represent "1" since 2nd reference bit is 1.

5.2 Evaluation and Experiments Settings

In order to compare the respective method (DCAXCSR, and XCSR), we use the correct rate of classification of XCSR as evaluation criteria. The learning of XCSR conducted 10 times for each method, and result is shown the average value of 10 trials. When the output of system is correct, the environment gives reward of 1000, and 0 when incorrect. The parameter of each method are set to indicated values of Table 1. Note that the parameter of "DCA: hidden layers structure" in the table indicates the first half layers structure because the hidden layers structure of DCA is symmetry. DCA compresses 784 dimensions to 4 dimensions in the MNIST dataset, and 11 dimensions to 9 dimensions in the 11-multiplexer problem.

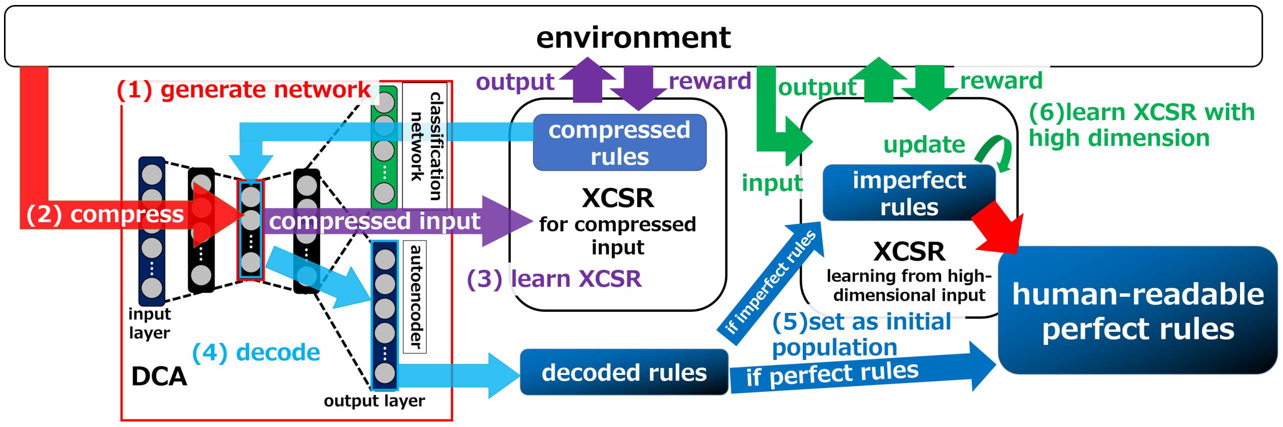

Figure 5: An Algorithm Flow of DCAXCSR

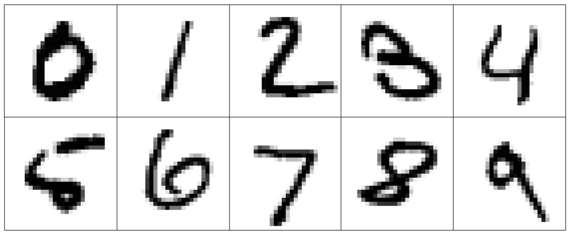
Figure 6: Examples of MNIST Dataset

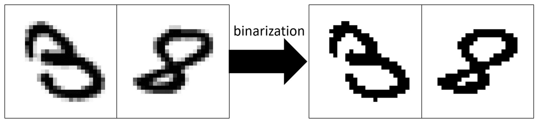

Figure 7: Examples of Binarized Images

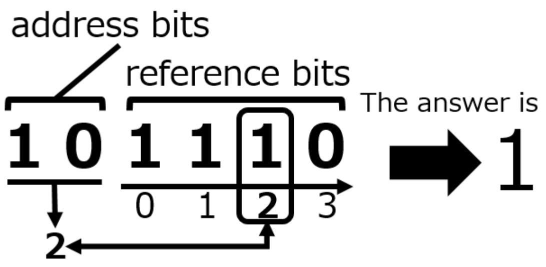

Figure 8: An Example of 6-multiplexer Problem

"XCSR: *N*" is the maximum population size of XCSR which is number maximum classifier XCSR can have. Other parameters of XCSR of each method are set to following [16]. If the correct rate of XCSR for compressed input is low, set the decoded classifiers to the XCSR learning from dimensional input, and learn it (Corresponding to (5)-(6) in section 4.3). In this section, result of "XCSR for compressed input" is denoted as "DCAXCSR (without knowledge) " which is standing for "DCAXCSR without using compression rules as knowledge", and result of "XCSR learning from high-dimensional input" is denoted as "DCAXCSR (with knowledge)" which is standing for "DCAXCSR with using compression rules as knowledge".

Table 1: Parameters of Experiments

parameter	MNIST	MUX
compression dimensions	784 ⟶ 4	11 ⟶ 9
DCA: training epoch	150000	300000
DCA: batch size	100	100
DCA: hidden layers structure	196-49-12-4	22-9
XCSR: training iteration	50000	50000
XCSR: *N*	1000	1000

5.3 Result

5.3.1 The MNIST Dataset. Figure 9 shows the correct rate of classification of "3" or "8" of The MNIST dataset of DCAXCSR (without knowledge) and XCSR. The vertical axis and the horizontal axis indicate the correct rate of classification of XCSR (%), and learning iterations of XCSR. From this figure, we can see that against XCSR which cannot solve that the correct answer rate is around 50%, DCAXCSR (without knowledge) converges to nearly 100%. Since both of the MNIST dataset and the 11-multiplexer problem are to classify to 2 classes, note that 50% of the output will be correct if output is selected randomly.

Figure 10 shows examples of acquired rules of DCAXCSR. The left image of the figure shows the rule which argues "it is 8" and the right image of the figure shows the rule which argues "it is 3". Red pixels indicates the "don't care" which means the pixel can be either white or black. The acquired knowledge from the figure is that "if the indentation on the left side of '3' is buried by black pixels, then it is '8', else it is '3' ".

5.3.2 11-Multiplexer Problem. Figure 11 shows the correct rate of 11-multiplexer problem of DCAXCSR (without knowledge), DCAXCSR (with knowledge) and XCSR. The vertical axis and the horizontal axis indicate the correct rate of classification of XCSR

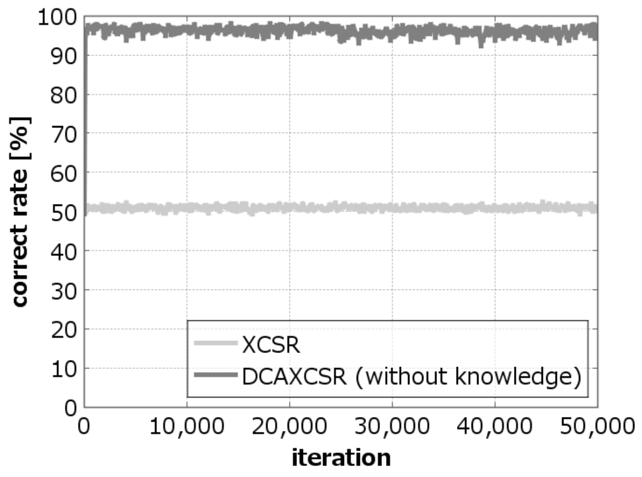

Figure 9: The Correct Rate of Classification of "3" or "8" of The MNIST Dataset

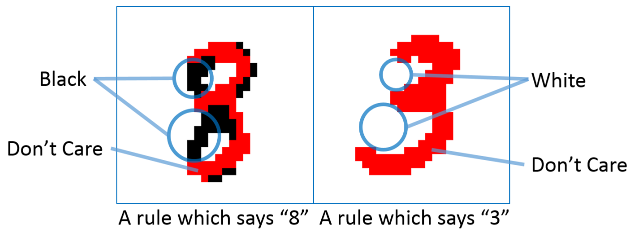

Figure 10: Examples of Acquired Rules of DCAXCSR

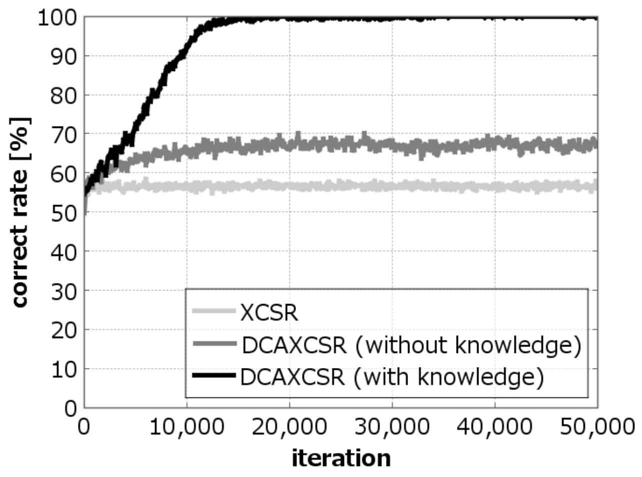

Figure 11: The Correct Rate of 11-Multiplexer Problem

(%), and learning iterations of XCSR. From this figure, the correct answer rate of XCSR converges to about 56%, DCAXCSR (without knowledge) converges to about 68% and DCAXCSR (with knowledge) converges to 100%. Thus we can say that XCSR cannot solve this problem, DCAXCSR (without knowledge) can hardly solve it, however, DCAXCSR (with knowledge) can solve it.

Table 2 shows top 3 examples which sorted by fitness of acquired rules in each XCSR. The symbol "#" of the table indicates don't care

Table 2: Examples of Acquired Rules

XCSR

	condition:action	p
O	01001001000 : 0	1000
O	11111110010 : 0	1000
O	00111101111 : 0	0

DCAXCSR (without knowledge)

	condition:action	p
X	#10##111111: 1	941
X	#01##010011: 0	645
X	#11##111111: 1	926

DCAXCSR (with knowledge)

	condition:action	p
O	1100110#00#: 1	0
O	0000#######: 1	0
O	100####0###: 1	0

which means either 0 or 1 is acceptable. The rule which has more number of "#" is more general because it can match many inputs. The leftmost column indicates whether or not the rule is correct. "O" indicates correct rule, and "X" indicates incorrect rule. From this table, we can see XCSR could obtain correct rules however they are not generalized, DCAXCSR without knowledge obtained some wrong rules, and DCAXCSR with knowledge obtained correct generalized rules.

5.4 Discussion

From Figure 9, we can say that it is too high-dimension for XCSR to learn MNIST dataset (784 dimensions) because the XCSR cannot solve it at all. However, DCAXCSR can solve it because inputs are compressed enough and correctly by DCA. In this problem, $N = 1000$ is too little for XCSR to solve, on the other hand, $N = 1000$ is enough for DCAXCSR because dimension is compressed to 1/196 size. There is a possibility for XCSR to solve it if N is increased significantly, however it cannot be solved in realistic calculation time. This means that DCAXCSR can solve the high-dimensional problem which can be compressed by neural network.

Figure 10 shows DCA of DCAXCSR can decode compressed rules correctly. Acquired rules (images) has common red "3" shape which means shape of "3" and "8" has a common shape of "3", and differ of them is whether the pixels of indentation on the left side of "3" is buried by black or while. We can say these are human-readable generalized rules.

From Figure 11, it is difficult for XCSR to learn from 11-multiplexer problem with $N = 1000$ because of the shortage of N. Incidentally, by increasing N enough, it was confirmed that it is possible even for XCSR to learn from 11-multiplexer problem. It is also difficult for neural network to learn this problem. Because input of multiplexer problem cannot be compressed since distribution of features are vary for each input. When compressing the input with DCA, multiple inputs may be mapped to the same data (overlap). In addition, since uncompressible input is compressed forcibly, necessary information may be lost. Using such incorrectly incompletely compressed input for XCSR generates rules with incorrect parts and cannot learn perfectly. Thus, DCAXCSR without knowledge cannot solve the problem perfectly. However, correct rate of DCAXCSR with knowledge converged to 100%. This is because compressed imperfect rules are used for narrowing down the state space of input. The calculation cost of modifying imperfect rules to perfect rules is smaller than generate perfect rules from nothing. This fact is seen in Table 2. There are partially wrong rules acquired by DCAXCSR without knowledge, in the table, 1st and 3rd rules have incorrect p and the 2nd rule has a wrong generalization, however the other part is correct. On the other hand, rules acquired by DCAXCSR with knowledge are correct. In addition, rules acquired by XCSR is not incorrect, however they are not generalized because there are no "#" in the rules, in contrast, rules acquired by DCAXCSR with knowledge are well-generalized.

From these reasons, we can say following implications; DCAXCSR can 1) learn from high-dimensional input and solve it; 2) learn difficult problems which DCA cannot learn and compress well by using compressed imperfect rules and 3) acquire well-generalized human-readable rules.

6 CONCLUSION

Learning Classifier System (LCS) is rule based machine learning which discovers knowledge from input. LCS can acquire human-readable generalized if-then rules as knowledge, however it is difficult for LCS to learn from high-dimensional input. To solve this problem, this paper proposed new LCS called DCAXCSR which can solve high-dimensional problems, and obtain human-readable generalized knowledge from high-dimensional environment. DCAXCSR is a hybrid system of XCSR which is one of LCS which can deal with continuous real numbers and DCA. DCA is dimension compressor with necessary features by combining the deep neural network for classification and autoencoder.

In order to investigate the effectiveness of DCAXCSR, this paper conducted experiments on the benchmark classification problem of MNIST database and 11-multiplexer problems. Samples of the MNIST database are images which has 784 dimensions. And 11-multiplexer problem is difficult problem because it is impossible for neural network to compress its input because distribution of features of input is vary for each input. DCA compress MINIST samples to 4 dimensions, and 11-multiplexer problems to 9 dimensions. The experiment results revealed following implements: DCAXCSR can 1) learn from high-dimensional input and solve it; 2) learn difficult problems which DCA cannot learn and compress well by using compressed imperfect rules and 3) acquire well-generalized human-readable rules.

Future work is that to solve more difficult various problems to verify applicability of DCAXCSR. And since the current system can not judge whether the rule is a perfect or imperfect only from the correct rate of classifiers, it is necessary to consider the judgment criteria to classify perfect rule or imperfect rule in the future.

REFERENCES

[1] Mani Abedini and Michael Kirley. 2011. *AI 2011: Advances in Artificial Intelligence: 24th Australasian Joint Conference, Perth, Australia, December 5-8, 2011. Proceedings.* Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Guided Rule Discovery in XCS for High-Dimensional Classification Problems, 1–10. https://doi.org/10.1007/978-3-642-25832-9_1

[2] Martin. V. Butz, David E. Goldberg, and Kurian Tharakunnel. 2003. Analysis and Improvement of Fitness Exploitation in XCS: Bounding Models, Tournament Selection, and Bilateral Accuracy. *Evolutionary Computation* 11, 3 (2003), 239–277.

[3] Garrison W Cottrell and Paul Munro. 1988. Principal components analysis of images via back propagation. In *Visual Communications and Image Processing'88: Third in a Series.* International Society for Optics and Photonics, 1070–1077.

[4] David E Golberg. 1989. Genetic algorithms in search, optimization, and machine learning. *Addion wesley* 1989 (1989).

[5] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (2006), 1527–1554.

[6] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.

[7] Muhammad Iqbal, Will N Browne, and Mengjie Zhang. 2014. Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. *Evolutionary Computation, IEEE Transactions on* 18, 4 (2014), 465–480.

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

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[10] Kazuma Matsumoto, Takato Tatsumi, Hiroyuki Sato, Tim Kovacs, and Keiki Takadama. 2017. XCSR Learning from Compressed Data Acquired by Deep Neural Network (Special Issue on Cutting Edge of Reinforcement Learning and its Applications). *Journal of advanced computational intelligence and intelligent informatics* 21, 5 (2017), 856–867.

[11] Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5, 4 (1943), 115–133.

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5 (1988), 3.

[13] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.

[14] Stewart W Wilson. 1995. Classifier fitness based on accuracy. *Evolutionary computation* 3, 2 (1995), 149–175.

[15] Stewart. W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3, 2 (June 1995), 149–175.

[16] Stewart W Wilson. 2000. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems.* Springer, 209–219.