Selective desensitization neural network

A1 Introduction

The selective desensitization neural network (SDNN) is a feedforward neural network comprising three parts: pattern coding (PC), selective desensitization (SD), and parallel perceptron (PP). Figure A1 shows the basic structure of the SDNN with N inputs. First, each input signal (feature value/symbol) is converted into multiple ($N-1$) binary patterns in the PC layer. Next, these binary patterns are integrated by desensitizing (i.e., neutralizing) a part of one pattern according to another pattern in the SD layer. The PP models a relationship between the desensitized patterns and the SDNN output.

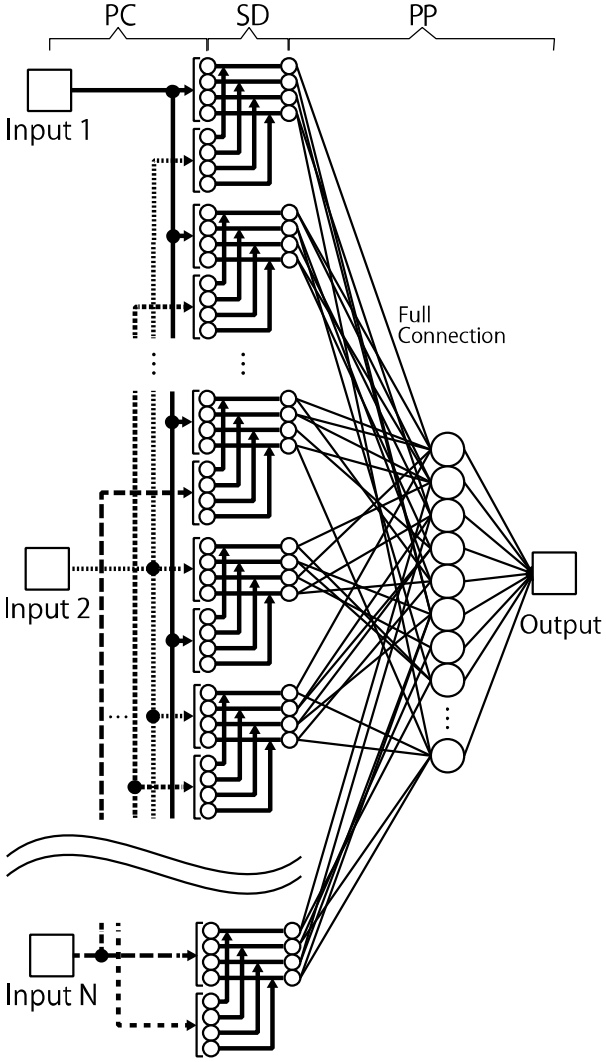

Figure A1: Structure of SDNN

A2 Pattern Coding

PC is a method for converting an input signal into a binary (i.e., +1 and −1) pattern, called "code pattern." PC enhances the expression ability (Tanno et al., 2015) and robustness against irrelevant input dimensions (Tanno et al., 2017) of a simple perceptron.

PC is implemented using the lookup-table method. The code patterns should have the characteristics presented in Table A1. In this library, we provide two methods, random-inverse and correlation-matrix methods, to create such code patterns, which are mainly for numerical and symbolic inputs, respectively.

(1) A code pattern should have the same number of +1 and −1 elements. Therefore, the number of elements in the code pattern should be even.
(2) Two code patterns should have high correlation if the corresponding two input signals are similar to each other. The correlation should decrease to nearly zero as the input signals become less similar and unrelated. Here, the correlation, $\mathrm{C}(\boldsymbol{P}_a, \boldsymbol{P}_b)$, between two patterns, $\boldsymbol{P}_a$ and $\boldsymbol{P}_b$, are defined as follows: $$\mathrm{C}(\boldsymbol{P}_a, \boldsymbol{P}_b) = \frac{\sum_{i=1}^{n} P_a(i) * P_b(i)}{n}.$$ Here, n represents the number of elements of $\boldsymbol{P}_a$ and $\boldsymbol{P}_b$, and $P_a(i)$ and $P_b(i)$ are the values of the i^{th} element of $\boldsymbol{P}_a$ and $\boldsymbol{P}_b$.

Table A1: Requirements for code patterns

A2-1 Random-Inverse Method

The random-inverse is used to create code patterns for numerical inputs, and also for symbolic inputs that can be placed on a number line.

First, this method randomly creates the code pattern for the minimum input value. Then, the other code patterns are created by flipping some elements of the neighboring patterns. The detailed procedure for numerical input is displayed in Algorithm A1.

Algorithm A1: Random-inverse method

Input values are quantized into q bins, and each bin is assigned to one n-dimensional binary (±1) code pattern (n is even). The code patterns, $\boldsymbol{P}_1, \cdots, \boldsymbol{P}_q$, are created as follows:

1. $\boldsymbol{P}_1$, the code pattern of the first bin, is created by setting half randomly selected elements to 1 and the remaining half to −1.
   r elements with +1 and −1 are randomly selected from $\boldsymbol{P}_1$, and $\boldsymbol{P}_2$ is created by flipping the signs of these $2r$ elements so that the number of +1 and −1 elements remains unchanged.
2. Similarly, $\boldsymbol{P}_k$ is created by flipping the signs of $2r$ elements in $\boldsymbol{P}_{k-1}$.

Three parameters, n, q, and r, exist in the above procedure, and their values have some effects on the performance of SDNN. n and q are directly related to the expression ability of the SDNN. Larger values of n and q are always preferable if we disregard computational costs (Nonaka et al., 2011). r is related to the generalization range and must satisfy $0 < r < n/4$ so that the correlation between the neighborhood patterns will not be 0 or less. $r = 1$ (or a small number) and $n \gg rq$ is recommended, because the small r allows training effects to spread farther, whereas the large n allows the SDNN output to vary rapidly.

By this method, the correlations between the two successive patterns are constant, and as the two input values grow farther apart, the correlation of their code patterns exponentially approaches 0 (Figure A2).

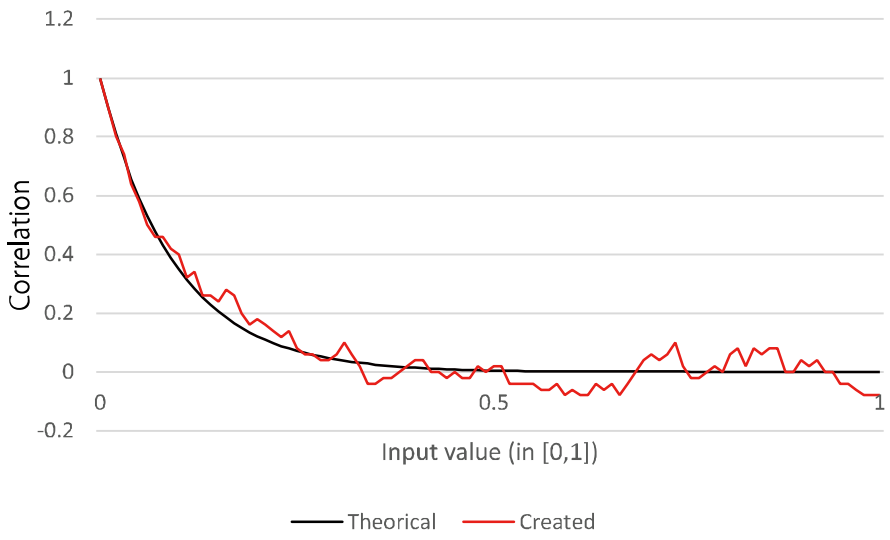

Figure A2: Correlation between the patterns assigned to 0 and the others
(Random-inverse, $\boldsymbol{n = 200}$, $\boldsymbol{q = 100}$, $\boldsymbol{r = 5}$)

A2-2 Correlation-Matrix Method

If the input symbols cannot be placed on a number line, the random-inverse method is not applicable. In addition, the random-inverse method is unsuitable for some numerical inputs. For example, when the input values represent an angle (radian), 0 and 2π denote the same angle but are encoded into uncorrelated patterns.

We developed a new PC method called the correlation-matrix method for such inputs. This method creates a list of code patterns according to a given matrix representing correlations between pairs of code patterns. This matrix can be designed freely according to the similarity between the input values/symbols if its elements satisfy the conditions presented in Table A2.

(1)	The matrix has to be square and symmetric.
(2)	The diagonal elements of the matrix have to be 1.
(3)	All elements must be in the range of $[-1, 1]$.
(4)	For all the combinations of three input values/symbols ($x, y,$ and z), the pattern correlations, $c_{x,y}, c_{y,z}$ and $c_{z,x}$, have to satisfy the following condition: $1 - \lvert c_{x,y} - c_{y,z} \rvert \geq c_{z,x} \geq \lvert c_{x,y} + c_{y,z} \rvert - 1.$ Refer also to Appendix A-4.

Table A12: Conditions to be satisfied by a correlation matrix

The correlation matrix method creates multiple "sub" code patterns and then makes each code pattern by concatenating these sub-code patterns. The detailed procedure is displayed in Algorithm A2.

Algorithm A2: Correlation-matrix method

We demonstrate creating q code patterns, $\boldsymbol{P}_1, \cdots, \boldsymbol{P}_q$, with n elements so that the correlation matrix for $(\boldsymbol{P}_1, \cdots, \boldsymbol{P}_q)$ will be as close as possible to the designated matrix:

$$\hat{C} = \begin{pmatrix} 1 & c_{1,2} & \cdots & c_{1,q} \\ c_{2,1} & 1 & & c_{2,q} \\ \vdots & & \ddots & \vdots \\ c_{q,1} & c_{q,2} & \cdots & 1 \end{pmatrix}.$$

Here, we define the sub-code patterns for the i^{th} code pattern, $\boldsymbol{P}_i$, as $\boldsymbol{P}_i^1, \cdots, \boldsymbol{P}_i^s$ where s is the number of sub-code patterns for each code pattern. The number of elements in the sub-code pattern is n/s and must be even.

1 The sub-code-patterns are created for each set of sub-code patterns with the same index. The creation procedure of the j^{th} sub-code-patterns, $\boldsymbol{P}_1^j, \cdots, \boldsymbol{P}_q^j$, is as follows:

1.1 Each sub-code pattern is created in a random order. Here, the first, second, and k^{th} $(k \geq 3)$ selected sub-code patterns are defined as $\boldsymbol{P}_{o_1}^j$, $\boldsymbol{P}_{o_2}^j$, and $\boldsymbol{P}_{o_k}^j$, respectively.

1.2 $\boldsymbol{P}_{o_1}^j$ is set by selecting $+1$ and -1 randomly for each element so that half of the elements takes $+1$ and the other half takes -1.

1.3 For $\boldsymbol{P}_{o_2}^j$, a temporary pattern, $\boldsymbol{P}_t$, is first set randomly in the same way as 1.2.

1.4 Calculate r as follows:

$$r = \left\lfloor \frac{n}{2} \cdot \left| \mathrm{C}\left(\boldsymbol{P}_{o_1}^j, \boldsymbol{P}_t\right) - c_{o_1,o_2} \right| + 0.5 \right\rfloor.$$

r represents the number of elements with +1 or −1 whose sign must be flipped.

1.4.1. If $\mathrm{C}\left(\boldsymbol{P}_{o_1}^j, \boldsymbol{P}_t\right) > c_{o_1,o_2}$, r elements with +1 and r elements with −1 are randomly selected from the elements in $\boldsymbol{P}_t$ whose values are the same as those in $\boldsymbol{P}_{o_1}^j$.

1.4.2. If $\mathrm{C}\left(\boldsymbol{P}_{o_1}^j, \boldsymbol{P}_t\right) < c_{o_1,o_2}$, r elements with +1 and r elements with −1 are randomly selected from the elements in $\boldsymbol{P}_t$ whose values are different from those in $\boldsymbol{P}_{o_1}^j$.

Finally, $\boldsymbol{P}_{o_2}^j$ is created by flipping the signs of $2r$ selected elements in $\boldsymbol{P}_t$.

1.5 More than a third of selected sub-code patterns are created taking a precision parameter into account.

To create the k^{th} selected sub-code patterns, $\boldsymbol{P}_{o_k}^j$, a temporary pattern, $\boldsymbol{P}_t$, is first set randomly in the same manner as 1.2 and then updated for the sub-code patterns, $\boldsymbol{P}_{o_1}^j, \cdots, \boldsymbol{P}_{o_{k-1}}^j$, in the same manner as 1.4. This procedure is repeated until the following condition is satisfied:

$$\sqrt{\frac{\sum_{x=1}^{k-1} \left(\mathrm{C}\left(\boldsymbol{P}_{o_x}^j, \boldsymbol{P}_t\right) - c_{o_x,o_k}\right)^2}{k-1}} < \mathrm{pr}.$$

Here, pr is the precision parameter (one of the hyperparameters). Note that if this condition is not satisfied after the above procedure is repeated a specified number of times, pr needs to be reset to a slightly higher value, and $\boldsymbol{P}_t$ needs to be updated again.

Finally, the final temporary pattern, $\boldsymbol{P}_t$, is set as $\boldsymbol{P}_{o_k}^j$.

2 The code patterns, $\boldsymbol{P}_1, \cdots, \boldsymbol{P}_q$, are created by concatenating the sub-code patterns, $\boldsymbol{P}_1^1, \cdots, \boldsymbol{P}_1^s$, $\boldsymbol{P}_2^1, \cdots, \boldsymbol{P}_2^s$, $\cdots$, and $\boldsymbol{P}_q^1, \cdots, \boldsymbol{P}_q^s$, respectively.

Four parameters, $n, q, s,$ and pr, exist in the above procedure. n, q have the same role as the random-inverse method. s is the number of sub-code patterns for each code pattern. A very large or very small s leads to an increase in error between the created and ideal correlations. Thus, we often use a value close to $\sqrt{n}$ as s. pr is the end condition for the creation of the sub-code patterns and should be small. Note that a very small value of pr considerably increases the amount of calculation.

Figure A3 displays the correlation between the code patterns created according to the correlation matrix $\hat{C}$ shown in equation A1. In this example, $\hat{C}$ is a square matrix of order 100 and is designed to be suitable for an angle input normalized in the range of [0,1] in which the minimum and maximum values of angle inputs are similar.

$$\hat{C} = \begin{pmatrix} 1 & c_{1,2} & \cdots & c_{1,100} \\ c_{2,1} & 1 & & c_{2,100} \\ \vdots & & \ddots & \vdots \\ c_{100,1} & c_{100,2} & \cdots & 1 \end{pmatrix} \tag{A1}$$

$$c_{ij} = 0.9^{\mathrm{f}(i,j)}, \mathrm{f}(i,j) = \min(\|i-j\|, 100 - \|i-j\|)$$

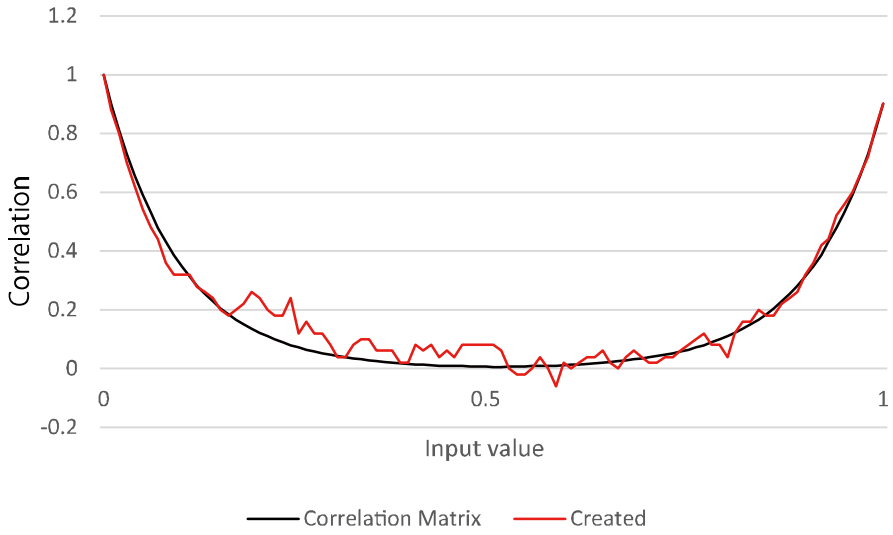

Figure A3: Correlation between the pattern assigned to 0 and the others

This method can enhance the accuracy and efficiency of modeling if the similarities among the input values/symbols are known. For example, consider the symbol inputs depicting the different types of animals (e.g., cats, dogs, and elephants). Considering the animals as pets, cats are similar to dogs but not elephants. Their code patterns can be designed such that those for cats and dogs are correlated and those for cats (or dogs) and elephants are uncorrelated using the correlation matrix method. This makes it easier for the SDNN to output similar values for cats and dogs.

A3 Selective Desensitization

SD is a method to integrate two code patterns by neutralizing the value of some elements of the pattern according to another pattern, where the desensitized pattern will be a ternary (+1, 0, −1) pattern (Figure A4; Morita et al., 2005).

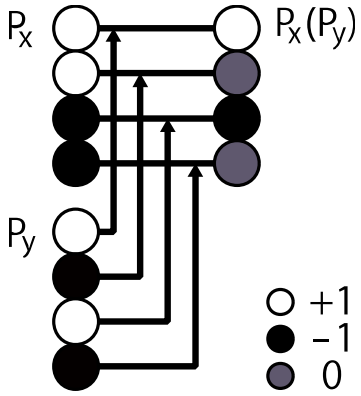

Figure A4: Example of selective desensitization

Specifically, consider two code patterns, $\boldsymbol{P}_x = (P_x(1), P_x(2), \cdots, P_x(n))$ and $\boldsymbol{P}_y = \big(P_y(1), P_y(2), \cdots, P_y(n)\big)$. The simplest method of SD is neutralizing $P_x(i)$ if $P_y(i) = -1$. That is, $\boldsymbol{P}_x$ is modified with $\boldsymbol{P}_y$ into the following equation:

$$\boldsymbol{P}_x\big(\boldsymbol{P}_y\big) = \left(\frac{\big(1 + P_y(1)\big)}{2} P_x(1), \ldots, \frac{\big(1 + P_y(n)\big)}{2} P_x(n) \right). \qquad \text{(A2)}$$

In general, SD is conducted for all combinations of modified and modifier inputs. Therefore, $N(N-1)$ ternary patterns are created, where N is the number of inputs. This procedure is called "mutual desensitization." If the important combinations of inputs are known, unimportant combinations can be omitted and the amount of calculation can be reduced.

SD together with PC greatly enhances the expression ability of a simple perceptron. For example, it gains the ability to solve the generalized 2-input XOR problem, where $F(x_1, y_1) = F(x_2, y_2) = 0$ and $F(x_1, y_2) = F(x_2, y_1) = 1$. Consequently, each simple perceptron comprising SDNN can form complicated decision boundaries.

A4 Parallel Perceptron

PP is one of the feedforward neural networks and comprises some simple perceptrons.

When the code patterns of N inputs are mutually desensitized, the output o_k and the inner potential μ_k of the k^{th} unit in the full-connection layer are defined as follows:

$$\begin{aligned} \mu_k &= \sum_{i=1}^{N}\sum_{j=1}^{N}\{f(i,j)\boldsymbol{w}_k^{ij}\cdot\boldsymbol{P}_i(\boldsymbol{P}_j)\}+h_k, \\ o_k &= \mathrm{H}_0(\mu_k), \\ f(i,j) &= \begin{cases}0\ (i=j)\\ 1\ (i\neq j)\end{cases} \end{aligned} \tag{A3}$$

Here, $\boldsymbol{w}_k^{ij}$ is the synaptic weight of k^{th} unit for the desensitized pattern, $\boldsymbol{P}_i(\boldsymbol{P}_j)$, which represents that the i^{th} code pattern is desensitized by the j^{th} code pattern, h_k is a bias of the k^{th} unit, and $\mathrm{H}_0(x)$ is a Heaviside function, ($\mathrm{H}_0(0)=0$).

The final output of SDNN is decided using the outputs in the full-connection layer, depending on pattern classification or function approximation tasks.

A4-1 Pattern Classification

In the pattern classification task, each unit in the full-connection layer in the PP works as a two-class classifier. For multi-class classification tasks, the one-versus-rest or one-versus-one method are prepared in *LIBSDNN*. Here, we consider an N_c-way multi-class classification.

In the one-versus-rest method, each unit corresponds to a class and classifies the corresponding and other classes. Thus, there are N_c elements in the full-connection layer. During the decision process, the class corresponding to the unit with the highest inner potential is selected as the final output.

In the one-versus-one method, each unit corresponds to a pair of classes (thus, there are ${}_{N_c}C_2$ units), and calculates to which class the inputs are likely to belong. In the decision process, the class chosen by the most units is selected as the final output.

The optimization of the synaptic weights and biases are conducted by error correction learning. The synaptic weights for the desensitized pattern, $\boldsymbol{P}_i(\boldsymbol{P}_j)$, and the bias of the k^{th} unit are optimized as follows:

$$\begin{aligned} \boldsymbol{w}_k^{ij} &\leftarrow \boldsymbol{w}_k^{ij}+\mathrm{sgn}(t_k-o_k)\cdot\boldsymbol{P}_i(\boldsymbol{P}_j), \\ h_k &\leftarrow h_k+\mathrm{sgn}(t_k-o_k). \end{aligned} \tag{A4}$$

Here, t_k is the target value of the k^{th} unit and $\mathrm{sgn}(x)$ is the signum function.

A4-2 Function Approximation

In the function approximation task (i.e., the final output), which is the approximated value of the target function, is calculated by $an_{+1}+b$, where a and b are constant, and n_{+1} is the number of units that output 1.

The training is achieved by error correction learning: the p-delta rule (P. Auer et al., 2008). The procedure is displayed in Algorithm A3.

Algorithm A2: p-delta rule

1 Calculate the number of units that should be updated (n_Δ) as follows:

$$n_\Delta=\left\lfloor\frac{|t-o|}{a}+0.5\right\rfloor.$$

Here, t and o are the values of the target function and final output, respectively.

1.1 If $t>o$, select n_Δ units outputting 0, in order, from internal potential closest to 0.

1.2 If $t<o$, select n_Δ units outputting 1, in order, from internal potentials closest to 0.

2 Train the n_Δ selected units using the error correction learning method (equation A4).

A5 Condition of the Correlation Matrix

The correlation matrix has to satisfy the conditions shown in Table A2 to make code patterns available for SDNN. In this section, we explain why the fourth condition is required (Table A2 (4)).

Depending on combinations of pattern correlations, there are cases where the code patterns with such correlations do not exist. Here, to obtain such conditions, we consider the range of $C(\boldsymbol{P}_z, \boldsymbol{P}_x)$ when $C(\boldsymbol{P}_x, \boldsymbol{P}_y)$ and $C(\boldsymbol{P}_y, \boldsymbol{P}_z)$ are determined, given three code patterns, $\boldsymbol{P}_x, \boldsymbol{P}_y, \boldsymbol{P}_z$, and correlations, $C(\boldsymbol{P}_x, \boldsymbol{P}_y)$, $C(\boldsymbol{P}_y, \boldsymbol{P}_z)$, $C(\boldsymbol{P}_z, \boldsymbol{P}_x)$ between them.

First, we define $\boldsymbol{D}_{a,b}$ as follows:

$$D_{a,b}(i) = \begin{cases} 1 & (P_a(i) = P_b(i)) \\ 0 & (otherwise) \end{cases}. \tag{A5}$$

Here, $D_{a,b}(i)$, $P_a(i)$, and $P_b(i)$ are the i^{th} elements of $\boldsymbol{D}_{a,b}$, $\boldsymbol{P}_a$, and $\boldsymbol{P}_b$, respectively. Each element of $\boldsymbol{D}_{a,b}$ represents whether $P_a(i)$ equals $P_b(i)$. For example, $\boldsymbol{D}_{a,a}$ is a vector whose elements are all 1.

We also define $m_{a,b}$ as the number of elements of $\boldsymbol{D}_{a,b}$ representing 1. Using $m_{a,b}$, the pattern correlation $C(\boldsymbol{P}_a, \boldsymbol{P}_b)$ can also be calculated as follows:

$$C(\boldsymbol{P}_a, \boldsymbol{P}_b) = \frac{2 * m_{a,b}}{n} - 1. \tag{A6}$$

Here, n is the total number of elements in the code patterns.

Now, consider the range of $m_{z,x}$, when $m_{x,y}$ and $m_{y,z}$ are determined. We assume that $\boldsymbol{D}_{x,y}$ is a vector whose elements are separated as follows:

$$\boldsymbol{D}_{x,y} = (0,0,\cdots,0,1,1,\cdots,1). \tag{A7}$$

Note that the number of elements with 1 is $m_{x,y}$.

$\boldsymbol{D}_{z,x}$ can be determined from the arrangement of the elements of $\boldsymbol{D}_{y,z}$. When $\boldsymbol{D}_{y,z} = (0,0,\cdots,0,1,1,\cdots,1)$, $m_{z,x}$ becomes maximal. When $m_{x,y} \geq m_{y,z}$,

$$\begin{aligned} \boldsymbol{D}_{x,y} &= (\underbrace{0 \quad \cdots \quad 0}_{n-m_{x,y}} \quad \underbrace{1 \quad \cdots \quad 1}_{m_{x,y}} \quad 1 \quad \cdots \quad 1) \\ \boldsymbol{D}_{y,z} &= (\underbrace{0 \quad \cdots \quad 0 \quad 0 \quad \cdots \quad 0}_{n-m_{y,z}} \quad \underbrace{1 \quad \cdots \quad 1}_{m_{y,z}}) \\ \boldsymbol{D}_{z,x} &= (\underbrace{1 \quad \cdots \quad 1}_{n-m_{x,y}} \quad 0 \quad \cdots \quad 0 \quad \underbrace{1 \quad \cdots \quad 1}_{m_{y,z}}) \end{aligned} \tag{A8}$$

Thus, maximum $m_{z,x}$ is as follows:

$$\max(m_{z,x}) = n + m_{y,z} - m_{x,y}. \tag{A9}$$

When $m_{x,y} < m_{y,z}$,

$$\begin{aligned} \boldsymbol{D}_{x,y} &= (\underbrace{0 \quad \cdots \quad 0 \quad 0 \quad \cdots \quad 0}_{n-m_{x,y}} \quad \underbrace{1 \quad \cdots \quad 1}_{m_{x,y}}) \\ \boldsymbol{D}_{y,z} &= (\underbrace{0 \quad \cdots \quad 0}_{n-m_{y,z}} \quad \underbrace{1 \quad \cdots \quad 1 \quad 1 \quad \cdots \quad 1}_{m_{y,z}}) \\ \boldsymbol{D}_{z,x} &= (\underbrace{1 \quad \cdots \quad 1}_{n-m_{y,z}} \quad 0 \quad \cdots \quad 0 \quad \underbrace{1 \quad \cdots \quad 1}_{m_{x,y}}) \end{aligned} \tag{A10}$$

Thus, the maximum $m_{z,x}$ is as follows:

$$\max(m_{z,x}) = n + m_{x,y} - m_{y,z}. \tag{A11}$$

Finally, the maximum $m_{z,x}$ is obtained from equations A9 and A11 as follows:

$$\max(m_{z,x}) = n - |m_{x,y} - m_{y,z}|. \tag{A12}$$

Conversely, when $\boldsymbol{D}_{y,z} = (1,1,\cdots,1,0,0,\cdots,0)$, $m_{z,x}$ becomes minimal. When $m_{x,y} + m_{y,z} \geq n$,

$$\begin{aligned}
\boldsymbol{D}_{x,y} &= (\underbrace{0 \quad \cdots \quad 0}_{n-m_{x,y}} \quad \underbrace{1 \quad \cdots \quad 1}_{m_{x,y}} \quad 1 \quad \cdots \quad 1) \\
\boldsymbol{D}_{y,z} &= (\underbrace{1 \quad \cdots \quad 1}_{m_{y,z}} \quad 1 \quad \cdots \quad 1 \quad \underbrace{0 \quad \cdots \quad 0}_{n-m_{y,z}}) \\
\boldsymbol{D}_{z,x} &= (\underbrace{0 \quad \cdots \quad 0}_{n-m_{x,y}} \quad 1 \quad \cdots \quad 1 \quad \underbrace{0 \quad \cdots \quad 0}_{n-m_{y,z}})
\end{aligned} \tag{A13}$$

The minimum $m_{z,x}$ is as follows:

$$\min(m_{z,x}) = m_{x,y} + m_{y,z} - n \tag{A14}$$

When $m_{x,y} + m_{y,z} < n$,

$$\begin{aligned}
\boldsymbol{D}_{x,y} &= (\underbrace{0 \quad \cdots \quad 0}_{n-m_{x,y}} \quad 0 \quad \cdots \quad 0 \quad \underbrace{1 \quad \cdots \quad 1}_{m_{x,y}}) \\
\boldsymbol{D}_{y,z} &= (\underbrace{1 \quad \cdots \quad 1}_{m_{y,z}} \quad 0 \quad \cdots \quad 0 \quad 0 \quad \cdots \quad 0) \\
\boldsymbol{D}_{z,x} &= (\underbrace{0 \quad \cdots \quad 0}_{m_{y,z}} \quad 1 \quad \cdots \quad 1 \quad \underbrace{0 \quad \cdots \quad 0}_{m_{x,y}})
\end{aligned} \tag{A15}$$

The minimum, $m_{z,x}$, is as follows:

$$\min(m_{z,x}) = n - (m_{x,y} + m_{y,z}) \tag{A16}$$

Finally, the minimum, $m_{z,x}$, is obtained from equations A14 and A16 as follows:

$$\min(m_{z,x}) = |m_{x,y} + m_{y,z} - n|. \tag{A17}$$

From equations A12 and A17, the range of $m_{z,x}$ is as follows:

$$n - |m_{x,y} - m_{y,z}| \geq m_{z,x} \geq |m_{x,y} + m_{y,z} - n|. \tag{A18}$$

Hence, using equations A6 and A18, the range of $C(\boldsymbol{P}_z, \boldsymbol{P}_x)$ can be obtained as follows:

$$\begin{aligned}
1 - |C(\boldsymbol{P}_x, \boldsymbol{P}_y) - C(\boldsymbol{P}_y, \boldsymbol{P}_z)| &\geq C(\boldsymbol{P}_z, \boldsymbol{P}_x) \\
&\geq |C(\boldsymbol{P}_x, \boldsymbol{P}_y) + C(\boldsymbol{P}_y, \boldsymbol{P}_z)| - 1.
\end{aligned} \tag{A19}$$

This equation is the condition in which the code patterns having the combination of the specified pattern correlations exist. Thus, if the correlation matrix that does not satisfy the fourth condition in Table A2 is set, *LIBSDNN* cannot make the code patterns.

References

T. Tanno, K. Horie, T. Kobayashi, and M. Morita. Effect of pattern coding on pattern classification neural network. *International Journal of Machine Learning and Computing*, 5(4): 339–343, 2015.
DOI: 10.7763/IJMLC.2015.V5.531.

T. Tanno, K. Horie, J. Izawa, and M. Morita. Robustness of selective desensitization perceptron against irrelevant and partially relevant features in pattern classification. In *International Conference on Neural Information Processing (ICONIP) 2017*: 520–529, 2017.
DOI: 10.1007/978-3-319-70136-3_55.

K. Nonaka, F. Tanaka, and M. Morita. Empirical comparison of feedforward neural network on two-variable function approximation (in Japanese). *IEICE TRANSACTIONS on Information and Systems*, J94(12): 2,114–2,125, 2011.

M. Morita, K. Matsuzaka, and S. Morokami. A model of context-dependent association using selective desensitization of nonmonotonic neural elements. *Systems and Computers in Japan*, 36(7): 73-83, 2005.
DOI: 10.1002/scj.10477.

P. Auer, H. Burgsteiner, and W. Maass. Learning rule for very simple universal approximators consisting of a single layer of perceptrons, *Neural Networks*, 21(5): 786–795, 2008.
DOI: 10.1016/j.neunet.2007.12.036