New Generation Computing (2024) 42:203–216
https://doi.org/10.1007/s00354-024-00256-3

NEW GENERATION COMPUTING

RESEARCH PAPER

Networks of Classical Conditioning Gates and Their Learning

Shun-ichi Azuma[1] · Dai Takakura[2] · Ryo Ariizumi[3] · Toru Asai[4]

Received: 5 July 2023 / Accepted: 22 March 2024 / Published online: 30 April 2024

Abstract
A research project on chemical AI, called the *Molecular Cybernetics Project*, was launched in Japan in 2021 with the goal of creating a molecular machine that can learn a type of conditioned reflex through the process of classical conditioning. In this project, we have developed a learning method for the network of such learning molecular machines, which is reported in this paper. First, as a model of a learning molecular machine, we formulate a logic gate that can learn conditioned reflex and introduce the network of the logic gates. Then we derive a key principle for learning, called the flipping principle, by which we present a learning algorithm for the network to realize a desired function.

Keywords Network · Learning · Classical conditioning · Pavlov's dog · Logic gate

✉ Shun-ichi Azuma
sazuma@i.kyoto-u.ac.jp

Dai Takakura
ultradait@gmail.com

Ryo Ariizumi
ryoariizumi@go.tuat.ac.jp

Toru Asai
asai@nuem.nagoya-u.ac.jp

1 Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan

2 Global Intellectual Property Group, Hitachi Ltd., 832-2, Horiguchi, Hitachinaka 312-0034, Japan

3 Faculty of Engineering, Tokyo University of Agriculture and Technology, 2-24-16 Naka-cho, Koganei 184-8588, Japan

4 Graduate School of Engineering, Chubu University, 1200 Matsumoto-cho, Kasugai 487-8501, Japan

1 Introduction

The development of DNA nanotechnology has raised expectations for *chemical AI*. Chemical AI is chemically synthesized artificial intelligence that has the ability of learning in addition to information processing. A research project on chemical AI, called the *Molecular Cybernetics Project*, was launched in Japan in 2021 [1], with the goal of establishing an academic field called *molecular cybernetics*.

One of the milestones of the project is to create a molecular machine that can learn a type of conditioned reflex through the process called *classical conditioning* [2]. Classical conditioning is the process of acquiring a conditioned reflex by giving an conditioned stimulus with an unconditioned stimulus. It was discovered by the well-known psychological experiment of "Pavlov's dog":

- if one feeds a dog repeatedly while ringing of a bell, then the dog will eventually begin to salivate at the sound of the bell, and
- if one just rings a bell repeatedly without doing anything else for the dog that salivates at the sound of the bell, the dog will stop salivating at the sound of the bell,

which are, respectively, called the *acquisition* and *extinction*. This project attempts to create liposomes with different functions and combine them to artificially achieve a function similar to classical conditioning.

If the milestone is achieved, the next step would be to configure a network of such machines to realize more complex functions. Therefore, it is expected to establish a learning method for such a network. However, there exists no learning method because the learning has to be performed by the interaction of classical conditioning on the network, which is completely different from what is being considered these days. In fact, neural networks are well known as a learning model, where learning is performed by adjusting weights between nodes [3], not by providing external inputs as in classical conditioning. On the other hand, Boolean networks [4] are known as a model of the network of logic gates and their learning has been studied, e.g., in [5], but it also differs from learning by providing external inputs. Moreover, there are a number of results on molecular computing, e.g., on molecular logic gates [7–10] and molecular computer [11], whereas they are not for classical conditioning.

In this paper, we develop a method for learning a desired function in a network of nodes, each of which can implement classical conditioning. First, classical conditioning is modeled as a time-varying logic gate with two inputs and single output. The two inputs correspond to the feeding and bell ringing in Pavlov's dog experiment, and the gate operates as either a projection gate or a logical OR gate at each time. The gate state, which is either projection or OR, is determined by how the inputs are given, in a similar manner to classical conditioning. The model is called here a *classical conditioning gate*. Based on this model, the network of classical conditioning gates and its learning problem are formulated. The "learning" considered here refers to obtaining a desired input–output relation of the network by steering the state of the network by applying an appropriate input sequence and thus what "learning" means is somewhat different from that used in the context of machine learning. For the learning, we use a key principle to solving the problem, called the *flipping principle*: that the gate state

Table 1 Truth tables of a projection function with respect to the first input and a logical OR

Input 1	Input 2	Output (projection)	Output (OR)
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	1

of any node in the network can be flipped while preserving the state of some other nodes. By the flipping principle, we present a learning algorithm to obtain a desired function on the network. It should be noted that the above project assumes that classical conditioning is implemented in liposomes, whereas the proposed learning method is applicable regardless of how classical conditioning is implemented.

Finally, we summarize the terminology and notation used in this paper. We consider two types of logical gates, a projection function with respect to the first input [6] and a logical OR. Table 1 shows the truth tables. We use $x_1 \vee x_2$ to represent the logical OR operation of the binary variables x_1 and x_2. Moreover, let $\bigvee_{i\in\mathbf{I}} x_i$ denote the logical OR operation of x_i with respect to all the indeces in a finite set $\mathbf{I}$. The composite function of two Boolean functions $h_1 : \{0,1\}^m \to \{0,1\}^n$ and $h_2 : \{0,1\}^n \to \{0,1\}^p$ is denoted by $h_2 \circ h_1$. Finally, we introduce the equivalence relation between two systems. Consider the systems S_1 and S_2, each of which is given by

$$S_i : \begin{cases} x_i(t+1) = f_i(x_i(t), u_i(t)), \\ y_i(t) = g_i(x_i(t), u_i(t)), \end{cases}$$

where $i \in \{1,2\}$. Here, $x_i(t) \in \{0,1\}^n$ is the state, $u_i(t) \in \{0,1\}^m$ is the input, and $y_i(t) \in \{0,1\}^p$ is the output of the system S_i, and $f_i : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$, and $g_i : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^p$ are functions. The system S_1 is said to be equivalent to S_2 or S_2 is said to be equivalent to S_1 if, for each $z_0 \in \{0,1\}^n$ and $(w_1, w_2, \ldots) \in \Pi_{j=0}^{\infty}\{0,1\}^m$, $x_1(t) = x_2(t)$ and $y_1(t) = y_2(t)$ hold for all $t \in \{0,1,\ldots\}$ under $x_1(0) = x_2(0) = z_0$ and $u_1(t) = u_2(t) = w_t$ $(t = 0, 1, \ldots)$.

2 System Modeling

As stated in Sect. 1, one of the milestones of the Chemical AI project is to create a molecular machine that mimics classical conditioning. Although the word "classical conditioning" has a broad notion in general, we focus here on one aspect of classical conditioning, as follows.

- It is a learning process of a system with two inputs and one output. The physical entity of the inputs and output are prespecified, and the inputs are classified as the primary and secondary inputs.
- If the output is correlated only with the primary input, it will also become correlated with the secondary input through learning process.

- If the output is correlated with both inputs, the output will become uncorrelated with the secondary input through learning process.

2.1 Classical Conditioning Gates

We model classical conditioning as shown in Fig. 1. It is a two-state machine that switches between the states "PRJ" and "OR" based on the two input signals taking a binary value. When the state is "PRJ", the gate operates as a projection function with respect to the first input, whose output is equal to the first input, as shown in Table 1. On the other hand, when the state is "OR", the gate operates as a logical OR gate, whose output is equal to 1 if and only if at least one of the two inputs is equal to 1. The state changes when two types of *training inputs* are applied. When the state is "PRJ", the state is changed to "OR" by entering the value (1, 1) several times in a row. On the other hand, when the state is "OR", the state is changed to "PRJ" by entering the value (0, 1) several times in a row.

This model can be interpreted in terms of Pavlov's dog experiment as follows. The state "PRJ" corresponds to responding only when the dog is being fed, while the state "OR" corresponds to responding when the dog is being fed or hears the bell. Then the input value (1, 1) is interpreted as the stimulus for acquisition, i.e., feeding with bell ringing, and (0, 1) is interpreted as the stimulus for extinction, i.e., bell ringing without feeding.

The model of the above classical conditioning gate is expressed as

$$\begin{cases} x(t+1) = \begin{cases} \mathrm{OR} & \text{if } x(t-s+1) = \mathrm{PRJ} \text{ and} \\ & \quad (v(\tau), w(\tau)) = (1,1)\ (\tau = t, t-1, \ldots, t-s+1), \\ \mathrm{PRJ} & \text{if } x(t-s+1) = \mathrm{OR} \text{ and} \\ & \quad (v(\tau), w(\tau)) = (0,1)\ (\tau = t, t-1, \ldots, t-s+1), \\ x(t) & \text{otherwise}, \end{cases} \\ y(t) = \begin{cases} v(t) & \text{if } x(t) = \mathrm{PRJ}, \\ v(t) \vee w(t) & \text{if } x(t) = \mathrm{OR}, \end{cases} \end{cases} \tag{1}$$

where $x(t) \in \{\mathrm{PRJ}, \mathrm{OR}\}$ is the state, $v(t) \in \{0, 1\}$ and $w(t) \in \{0, 1\}$ are the inputs, $y(t) \in \{0, 1\}$ is the output, and $s \in \{1, 2, \ldots\}$ is a period, called the *unit training time*. The state equation represents classical conditioning, while the output equation represents the resulting input–output relation at time t.

The following result presents a basic property of (1), which will be utilized for training a network of classical conditioning gates.

Lemma 1 *Consider the classical conditioning gate in (1) with $x(t) = \tilde{x}$, where $t \in \{0, 1, \ldots\}$ and $\tilde{x} \in \{\mathrm{PRJ}, \mathrm{OR}\}$ are arbitrarily given. Then the following statements hold.*

(i) If $(v(t), w(t)) = (0, 0)$, then $y(t) = 0$ and $x(t+1) = x(t)$.
(ii) If $(v(t), w(t)) = (1, 0)$, then $y(t) = 1$ and $x(t+1) = x(t)$.

Proof Trivial from (1). □

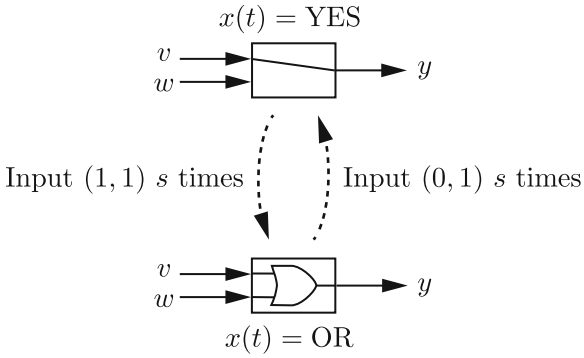

Fig. 1 Classical conditioning gate

This shows that there exists an input value that sets an arbitrary value at the output while preserving the state value.

2.2 Network of Classical Conditioning Gates

Now, we introduce a network of classical conditioning gates, as shown in Fig. 2. The network has a binary tree structure, where each gate, except for the leftmost gates, is connected to two other gates on the input side. The network has m layers, indexed by $1, 2, \ldots, m$ from the input side. The ith layer has 2^{m-i} gates, and thus the network has $\sum_{i=1}^{m} 2^{m-i}$ gates. We use n_i and n to denote these numbers, i.e., $n_i = 2^{m-i}$ $(i = 1, 2, \ldots, m)$ and $n = \sum_{i=1}^{m} 2^{m-i} = \sum_{i=1}^{m} n_i$.

We introduce the following notation for the network. The network with m layers is denoted by $\Sigma(m)$. The jth gate from the top in the i-layer is called *node* (i, j), and let $x_{ij}(t) \in \{\mathrm{PRJ}, \mathrm{OR}\}$, $v_{ij}(t) \in \{0, 1\}$, $w_{ij}(t) \in \{0, 1\}$, and $y_{ij}(t) \in \{0, 1\}$ be the state, first input, second input, and output of node (i, j), respectively. The pair of $v_{ij}(t)$ and $w_{ij}(t)$ is denoted by $u_{ij}(t) \in \{0, 1\}^2$. We use $\bar{x}_{ij}(t)$ to represent the flipped value of $x_{ij}(t)$: $\bar{x}_{ij}(t) = \mathrm{PRJ}$ for $x_{ij}(t) = \mathrm{OR}$, while $\bar{x}_{ij}(t) = \mathrm{OR}$ for $x_{ij}(t) = \mathrm{PRJ}$.

Next, let us consider the collection of signals. We use $\mathbf{N} := \{(1, 1), (1, 2), \ldots, (m, 1)\}$, which has n elements, to represent the set of node indeces, and use $\mathbf{N}_i \subset \mathbf{N}$ to represent the set of node indeces in the ith layer. Let $X_i(t) \in \{\mathrm{PRJ}, \mathrm{OR}\}^{n_i}$, $U_i(t) \in \prod_{j=1}^{n_i}(\{0, 1\} \times \{0, 1\})$, and $Y_i(t) \in \{0, 1\}^{n_i}$ be the collective state, input, and output of the ith layer. Let also $X(t) := (X_1(t), X_2(t), \ldots, X_m(t)) \in \{\mathrm{PRJ}, \mathrm{OR}\}^{n_1} \times \{\mathrm{PRJ}, \mathrm{OR}\}^{n_2} \times \cdots \times \{\mathrm{PRJ}, \mathrm{OR}\}^{n_m} = \{\mathrm{PRJ}, \mathrm{OR}\}^{n}$, $U(t) := U_1(t)$, and $Y(t) := Y_m(t)$. According to this notation, the state, input, and output of the network $\Sigma(m)$ are expressed as $X(t)$, $U(t)$, and $Y(t)$, respectively. Note that $Y(t) = Y_m(t) = y_{m1}(t)$.

Figure 3 shows an example for $m = 3$. In this case, we have $n_1 = 4$, $n_2 = 2$, $n_3 = 1$, $n = 7$, $\mathbf{N} = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (3, 1)\}$, $X_1(t) = (x_{11}(t), x_{12}(t), x_{13}(t), x_{14}(t))$, $X_2(t) = (x_{21}(t), x_{22}(t))$, $X_3(t) = x_{31}(t)$, $U_1(t) = (u_{11}(t), u_{12}(t), u_{13}(t), u_{14}(t))$, $U_2(t) = (u_{21}(t), u_{22}(t))$, $U_3(t) = u_{31}(t)$, $Y_1(t) = (y_{11}(t), y_{12}(t), y_{13}(t), y_{14}(t))$, $Y_2(t) = (y_{21}(t), y_{22}(t))$, $Y_3(t) = y_{31}(t)$, $X(t) = (x_{11}(t), x_{12}(t), x_{13}(t), x_{14}(t), x_{21}(t), x_{22}(t), x_{31}(t))$, $U(t) = (u_{11}(t), u_{12}(t), u_{13}(t), u_{14}(t))$, and $Y(t) = y_{31}(t)$.

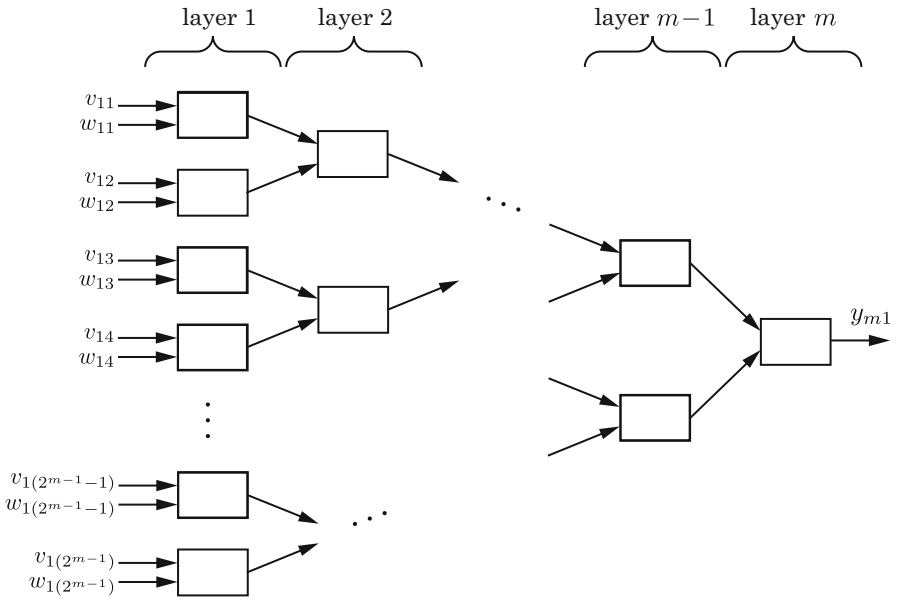

Fig. 2 Network of classical conditioning gates

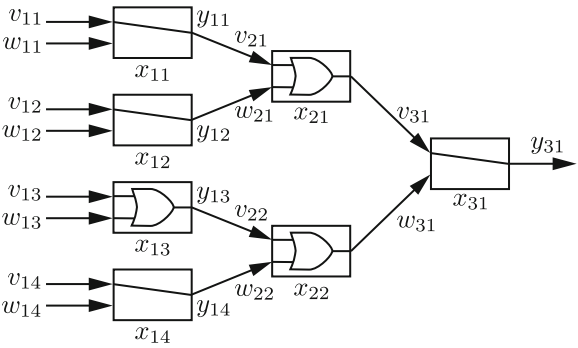

Fig. 3 Network $\Sigma(3)$ with $X(0) =$ (PRJ, PRJ, OR, PRJ, OR, OR, PRJ)

Since a classical conditioning gate operates as either a projection function or a logical OR, the possible input–output relation of $\Sigma(m)$ is limited to a logical OR of some of the inputs of $\Sigma(m)$. Moreover, the output $Y(t)$ always depends on the input $v_{11}(t)$, that is, there exists a function $g : \{0,1\}^{2n_1-1} \to \{0,1\}$ such that $Y(t) = v_{11}(t) \vee g(w_{11}(t), v_{12}(t), w_{12}(t), \ldots, v_{1(2^{m-1})}(t), w_{1(2^{m-1})}(t))$, because $v_{11}(t)$ propagates through nodes $(i,1)$ $(i = 1,2,\ldots,m)$ operating as a projection function or a logical OR. For example, $y_3(0) = v_{11}(0) \vee v_{12}(0)$ for the network $\Sigma(3)$ in Fig. 3. This fact is formalized as follows.

Lemma 2 *Consider the network $\Sigma(m)$ with $X(t) = \tilde{X}$, where $t \in \{0,1,\ldots\}$ and $\tilde{X} \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ are arbitrarily given. The following statements hold.*

(i) The input–output relation at time t is given by

$$Y(t) = \left(\bigvee_{j \in \mathbf{J}_1} v_{1j}(t) \right) \vee \left(\bigvee_{j \in \mathbf{J}_2} w_{1j}(t) \right) \tag{2}$$

for some $\mathbf{J}_1 \subseteq \{1, 2, \ldots, n_1\}$ and $\mathbf{J}_2 \subseteq \{1, 2, \ldots, n_1\}$, i.e., there exists a pair $(\mathbf{J}_1, \mathbf{J}_2)$ such that (2).

(ii) Consider the sets $\mathbf{J}_1$ and $\mathbf{J}_2$ satisfying (2). Then $1 \in \mathbf{J}_1$ holds. Moreover, if $v_{11}(t) = 1$, then $Y(t) = 1$. □

The following result shows a structural property that indicates which inputs affect node (i, j).

Lemma 3 *Consider the network $\Sigma(m)$ with $X(t) = \tilde{X}$ and any node $(i, j) \in \{2, 3, \ldots, m\} \times \{1, 2, \ldots, n_i\}$, where $t \in \{0, 1, \ldots\}$ and $\tilde{X} \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ are arbitrarily given. Let the input $U(t)$ be divided into $2n_i$ blocks of equal size and let $[U(t)]_k \in \{0, 1\}^{2^{i-1}}$ be the kth block. Then there exist functions $h_1 : \{0, 1\}^{2^{i-1}} \to \{0, 1\}$ and $h_2 : \{0, 1\}^{2^{i-1}} \to \{0, 1\}$ such that*

$$v_{ij}(t) = h_1([U(t)]_{2j-1}), \quad w_{ij}(t) = h_2([U(t)]_{2j}). \tag{3}$$

Proof It is trivial from the definition of $\Sigma(m)$. See Fig. 2. □

Lemma 3 is illustrated as follows. Consider the network $\Sigma(3)$ in Fig. 3 and node $(2, 1)$. Then $U(t)$ is divided into 4 blocks (where $2n_2 = 2^2$): $[U(t)]_1 = (v_{11}(t), w_{11}(t))$, $[U(t)]_2 = (v_{12}(t), w_{12}(t))$, $[U(t)]_3 = (v_{13}(t), w_{13}(t))$, and $[U(t)]_4 = (v_{14}(t), w_{14}(t))$. From Lemma 3, we have $v_{21}(t) = h_1([U(t)]_1)$ and $w_{21}(t) = h_2([U(t)]_2)$ for some $h_1 : \{0, 1\}^2 \to \{0, 1\}$ and $h_2 : \{0, 1\}^2 \to \{0, 1\}$. This is consistent with the interdependence between signals in Fig. 3.

3 Problem Formulation

In this paper, we aim at realizing a desired Boolean function on the network at a finite time T.

Consider node $(i, j) \in \mathbf{N}$ at time T. Let $f_{ij} : \{0, 1\}^2 \to \{0, 1\}$ be the input–output map of node (i, j) "at time T." By definition, f_{ij} is either a projection function with respect to the first input or a logical OR in the moment. Then the input–output map of layer i at time T, denoted by f_i, is expressed as

$$\begin{aligned} &f_i((v_{i1}(T), w_{i1}(T)), (v_{i2}(T), w_{i2}(T)), \ldots, (v_{i(2^{m-i})}(T), w_{i(2^{m-i})}(T))) \\ &= [f_{i1}(v_{i1}(T), w_{i1}(T)) \;\; f_{i2}(v_{i2}(t), w_{i2}(T)) \;\cdots \\ &\quad f_{i(2^{m-i})}(v_{i(2^{m-i})}(T), w_{i(2^{m-i})}(T)))]^\top, \end{aligned} \tag{4}$$

which allows us to represent the output of the network $\Sigma(m)$ at time T as

$$Y(T) = f_m \circ f_{m-1} \circ \cdots \circ f_1\big((v_{11}(T), w_{11}(T)), (v_{12}(T), w_{12}(T)), \ldots,$$

$$(v_{1(2^{m-i})}(T), w_{1(2^{m-i})}(T))\big). \tag{5}$$

By noting (5), our problem is formulated as follows.

problem 1 *Consider the network* $\Sigma(m)$ *with* $X(0) = \tilde{X}$*, where* $\tilde{X} \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ *is arbitrarily given. For each* $(i, j) \in \mathbf{N}$*, suppose that* $f_{ij} : \{0, 1\}^2 \to \{0, 1\}$ *is given as either a projection function with respect to the first input or a logical OR. Find a time* $T \in \{1, 2, \ldots\}$ *and an input sequence* $(U(0), U(1), \ldots, U(T-1)) \in \prod_{\tau=0}^{T-1} \{0, 1\}^{2n_1}$ *such that (5).* □

The following point should be noted for the problem. By the definitions of classical conditioning gates and networks, if the network is in the state $X(T) = X^*$ for an $X^* \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$, (5) holds for some functions $f_{11}, f_{12}, \ldots, f_{m1}$ that are either a projection function with respect to the first input or a logical OR gate. Conversely, if $f_{11}, f_{12}, \ldots, f_{m1}$ are arbitrarily given, there exists a (not necessarily unique) vector $X^* \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ such that $X(T) = X^*$ and (5) holds. For instance, such a vector is given by $X^* := (x^*_{11}, x^*_{12}, \ldots, x^*_{m1})$ for

$$x^*_{ij} := \begin{cases} \mathrm{PRJ} & \text{if } f_{ij} \text{ is a projection function with respect to the first input,} \\ \mathrm{OR} & \text{if } f_{ij} \text{ is a logical OR.} \end{cases} \tag{6}$$

Therefore, Problem 1 corresponds to finding an input sequence to steer the state from $X(0)$ to the terminal state X^* achieving (5).

4 Learning

Now, we present a solution to Problem 1.

4.1 Flipping Principle

Let us provide a key principle, called the *flipping principle*, for solving Problem 1.

The following is a preliminary result to derive the flipping principle.

Lemma 4 *Consider the network* $\Sigma(m)$ *with* $X(t) = \tilde{X}$*, where* $t \in \{0, 1, \ldots\}$ *and* $\tilde{X} \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ *are arbitrarily given. Then the following statements hold.*

(i) If $U(t) = (0, 0, \ldots, 0) \in \{0, 1\}^{2n_1}$*, then* $Y(t) = 0$ *and* $X(t+1) = X(t)$*. ss*
(ii) If $U(t) = (1, 0, \ldots, 0) \in \{0, 1\}^{2n_1}$*, then* $Y(t) = 1$ *and* $X(t+1) = X(t)$*.*

Proof In (i) and (ii), the relation $X(t+1) = X(t)$ is proven by the network structure of $\Sigma(m)$ and Lemma 1, which states that, in (1), $x(t+1) = x(t)$ holds under $(v(t), w(t)) = (0, 0)$ or $(v(t), w(t)) = (1, 0)$. Next, Lemma 2 (i) (in particular, (2)) implies that $Y(t) = 0$ for $U(t) = (0, 0, \ldots, 0)$, which proves (i). On the other hand, it follows from Lemma 2 (ii) that $Y(t) = 1$ for $U(t) = (1, 0, \ldots, 0)$. This proves (ii). □

Lemma 4 implies that there exists an input value for $\Sigma(m)$ that sets an arbitrary value at the output of $\Sigma(m)$ while preserving the state value. Note from this lemma that

the state of $\Sigma(m)$ does not change by an input sequence that takes $(0, 0, \ldots, 0)$ and $(1, 0, \ldots, 0)$ at each time.

From Lemma 4, we obtain the *flipping principle* for learning of $\Sigma(m)$.

Theorem 1 *Consider the network $\Sigma(m)$ with $X(t) = \tilde{X}$ and any node (p, q) in $\Sigma(m)$, where $t \in \{0, 1, \ldots\}$ and $\tilde{X} \in \{\mathrm{PRJ}, \mathrm{OR}\}^n$ are arbitrarily given. Then the following statements hold.*

(i) There exists an input sequence $(U_t, U_{t+1}, \ldots, U_{t+s-1}) \in \prod_{\tau=0}^{s} \{0, 1\}^{2n_1}$ such that

$$x_{ij}(t+s) = \begin{cases} \bar{x}_{ij}(t) & \text{if } (i, j) = (p, q), \\ x_{ij}(t) & \text{if } (i, j) \in \mathbf{N}_p \setminus \{(p, q)\} \end{cases} \tag{7}$$

under $U(t) = U_t$, $U(t+1) = U_{t+1}$, …, $U(t+s-1) = U_{t+s-1}$, where $s \in \{0, 1, \ldots\}$ is the unit training time defined for (1).

(ii) An input sequence satisfying (7) is given by $(\hat{U}_{(p,q)}, \hat{U}_{(p,q)}, \ldots, \hat{U}_{(p,q)})$ of length s (constant on the time interval $\{t, t+1, \ldots, t+s-1\}$), where $\hat{U}_{(p,q)} \in \{0, 1\}^{2n_1}$ is an input value which is divided into 2^{m+1-p} blocks of equal size, i.e., of size 2^{p-1} and whose blocks are given as follows:

$$\begin{aligned} (2q-1)\textit{th block} &: \begin{cases} (0, 0, 0, \ldots, 0) \text{ if } x_{pq}(t) = \mathrm{OR}, \\ (1, 0, 0, \ldots, 0) \text{ if } x_{pq}(t) = \mathrm{PRJ}, \end{cases} \\ 2q\textit{th block} &: (1, 0, 0, \ldots, 0), \\ \textit{Other blocks} &: (0, 0, 0, \ldots, 0). \end{aligned} \tag{8}$$

Proof Statements (i) and (ii) are proven by showing that (7) holds for the input sequence $(\hat{U}_{(p,q)}, \hat{U}_{(p,q)}, \ldots, \hat{U}_{(p,q)})$ of length s specified in (ii). The proof is given by dividing into two cases: $p = 1$ and $p \geq 2$.

If $p = 1$, it is trivial from (1), i.e., the definition of classical conditioning gates, that (7) holds for the input sequence specified in (ii).

Next, consider the case $p \geq 2$.

By definition, the network $\Sigma(m)$ can be represented as the cascade connection of m layers as shown in Fig. 4. As we can see by comparing Figs. 2 and 4, if $i \geq 2$, the entire left side of the ith layer is the parallel system of $2n_i$ subsystems, denoted by $S_1, S_2, \ldots, S_{2n_i}$, as shown in Fig. 5. Each subsystem is equivalent to the network of $i-1$ layers, i.e., $\Sigma(i-1)$, in the sense of the equivalence relation defined at the end of Sect. 1. This allows us to apply Lemma 4 to each subsystem because Lemma 4 holds for any $m \in \{1, 2, \ldots\}$.

Now, consider node (p, q). Suppose that $\Sigma(m)$ is represented as Fig. 5 for $i = p$, and let $Z_k(t) \in \{\mathrm{PRJ}, \mathrm{OR}\}^{\nu_p}$ be the state of the subsystem S_k, where $\nu_p := \sum_{i=1}^{p-1} 2^{i-1}$ and $\nu_p \geq 1$ because $p \geq 2$. By the definition of the subsystems S_k $(k = 1, 2, \ldots, 2n_p)$, the tuple $(Z_1(t), Z_2(t), \ldots, Z_{2n_p}(t))$ is the collective states of the nodes indexed in the set $\mathbf{N}_{p-1}$, from which the following statements are equivalent:

- $Z_k(t+s) = Z_k(t)$ for every $k \in \{1, 2, \ldots, 2n_p\}$.

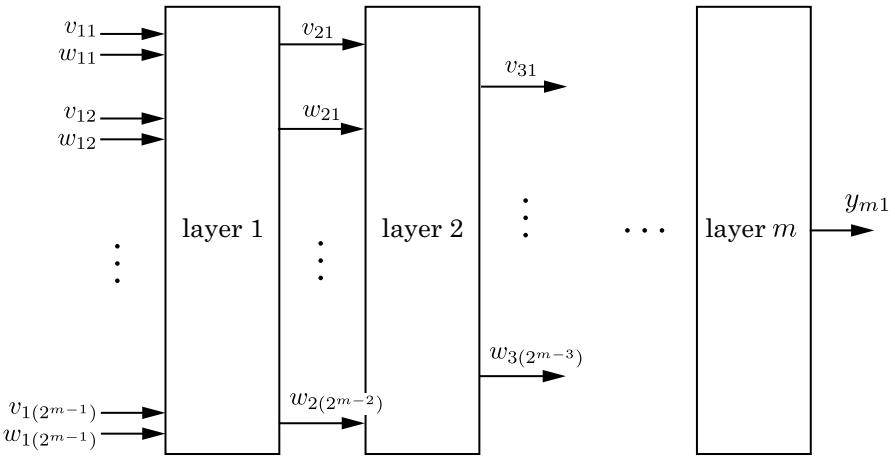

Fig. 4 Layer-based representation of network $\Sigma(m)$

- $x_{ij}(t+s) = x_{ij}(t)$ for every $(i, j) \in \mathbf{N}_{p-1}$.

Now, the proof is done for each of two cases: $x_{pq}(t) = \text{OR}$ and $x_{pq}(t) = \text{PRJ}$. First, we consider the case $x_{pq}(t) = \text{OR}$. If $x_{pq}(t) = \text{OR}$ and $U(t) = \hat{U}_{(p,q)}$, it follows from Lemmas 3 and 4 (Lemma 4 is applied to $\Sigma(p-1)$) that $(v_{pq}(t), w_{pq}(t)) = (0, 1)$, $(v_{pj}(t), w_{pj}(t)) = (0, 0)$ for $j \in \{1, 2, \ldots, n_p\} \setminus \{q\}$, and $Z_k(t+1) = Z_k(t)$ for $k \in \{1, 2, \ldots, 2n_p\}$. Thus if $U(t) = \hat{U}_{(p,q)}$, $U(t+1) = \hat{U}_{(p,q)}$, …, $U(t+s-1) = \hat{U}_{(p,q)}$, then

- $x_{pq}(t+s) = \text{PRJ} = \bar{x}_{pq}(t)$,
- $x_{pj}(t+s) = x_{pj}(t)$ for $j \in \{1, 2, \ldots, n_p\} \setminus \{q\}$,
- $Z_k(t+s) = Z_k(t)$ for $k \in \{1, 2, \ldots, 2n_p\}$.

This implies (7). The other case $x_{pq}(t) = \text{PRJ}$ can be proven in the same manner. The only difference is that $(v_{pq}(t), w_{pq}(t)) = (1, 1)$ holds when $x_{pq}(t) = \text{OR}$ and $U(t) = \hat{U}_{(p,q)}$. □

Theorem 1 implies that we can flip the state of any node while preserving the state of the other nodes in the layer to which the node to be flipped belongs and its upstream layers.

Example 1 Consider the network $\Sigma(3)$ in Fig. 3, where $X(0) = (\text{PRJ}, \text{PRJ}, \text{OR}, \text{PRJ}, \text{OR}, \text{OR}, \text{PRJ})$. By the input sequence $(\hat{U}_{(2,1)}, \hat{U}_{(2,1)}, \ldots, \hat{U}_{(2,1)})$ of length s for $\hat{U}_{(2,1)} = ((0, 0), (1, 0), (0, 0), (0, 0))$, the state of node $(2, 1)$ is flipped while preserving the states of nodes $(1, 1)$, $(1, 2)$, $(1, 3)$, $(1, 4)$, and $(2, 2)$. Figure 6 shows $\Sigma(3)$ with the resulting state $X(s)$. □

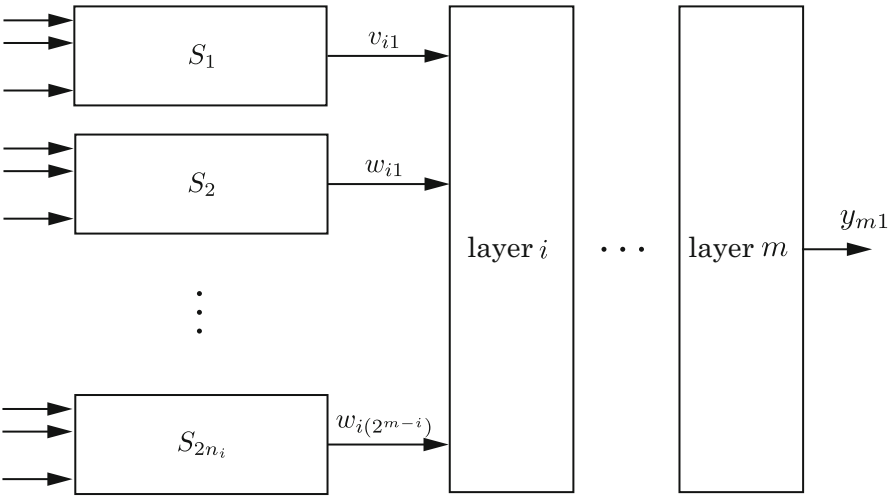

Fig. 5 Another layer-based representation of network $\Sigma(m)$

4.2 Learning Algorithm

Theorem 1 implies that we can steer the state of the network $\Sigma(m)$ from any value to any value by flipping the state of each node one by one from the upstream node. Based on this idea, we obtain the following algorithm to solve Problem 1.

Algorithm 1

(Step 1) For each $(i, j) \in \mathbf{N}$, let $x^*_{ij} \in \{\text{PRJ}, \text{OR}\}$ be defined by (6). Let also $k := 0$ and $\hat{\mathbf{N}} := \mathbf{N}$.

(Step 2) Pick the minimum pair (i, j) from $\hat{\mathbf{N}}$ in lexicographical order.

(Step 3) If $x_{ij}(ks) \neq x^*_{ij}$, apply the input sequence $(\hat{U}_{(i,j)}, \hat{U}_{(i,j)}, \ldots, \hat{U}_{(i,j)})$ of length s to the network $\Sigma(m)$, i.e., $U(ks) = \hat{U}_{(i,j)}, U(ks+1) = \hat{U}_{(i,j)}, \ldots, U(ks+s-1) = \hat{U}_{(i,j)}$, and let $k := k + 1$.

(Step 4) Let $\hat{\mathbf{N}} := \hat{\mathbf{N}} \setminus \{(i, j)\}$. If $\hat{\mathbf{N}} \neq \emptyset$, goto Step 2; otherwise, halt.

In the algorithm, k is a variable to count the number of nodes whose state is flipped, and $\hat{\mathbf{N}}$ is the list of the nodes for which the algorithm has never checked whether their state needs to be flipped or not. In Step 1, x^*_{ij} is defined for each $(i, j) \in \mathbf{N}$, where f_{ij} ($(i, j) \in \mathbf{N}$) are given functions in Problem 1. Moreover, k and $\hat{\mathbf{N}}$ are initialized. Step 2 picks a node (i, j) from the list $\hat{\mathbf{N}}$. Step 3 checks whether the state of node (i, j) has to be flipped or not; if it has to be flipped, the state is flipped by applying the training input sequence specified in Theorem 1. Note here that s steps of actual time elapsed while applying the input sequence to $\Sigma(m)$. In Step 4, node (i, j) is removed from the list $\hat{\mathbf{N}}$. In addition, the terminate condition is checked; if $\hat{\mathbf{N}}$ is empty, the algorithm terminates; otherwise, the above procedure is performed for a remaining node in the list $\hat{\mathbf{N}}$.

For the algorithm, we obtain the following result.

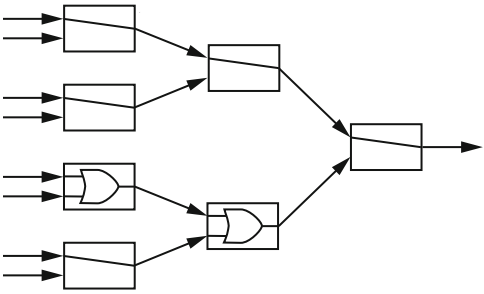

Fig. 6 Network $\Sigma(3)$ at the state $X(3)$, resulted by the input sequence $\mathbb{U}(0) = (\hat{U}_{(2,1)}, \hat{U}_{(2,1)}, \hat{U}_{(2,1)})$ that flipsthe state of node (2, 1)

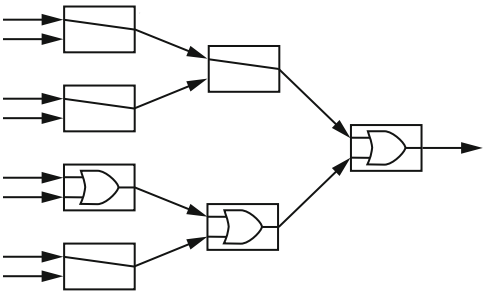

Fig. 7 Network $\Sigma(3)$ at the state $X(6)$, resulted by the input sequence $\mathbb{U}(1) = (\hat{U}_{(3,1)}, \hat{U}_{(3,1)}, \hat{U}_{(3,1)})$ that flips the state of node (3, 1)

Theorem 2 *Consider Problem 1. Let* $k^* \in \{0, 1, \ldots\}$ *be the value of* k *when Algorithm 1 terminates and* $\mathbb{U}(k)$ $(k = 0, 1, \ldots k^* - 1)$ *are the input sequence generated in Step 3 of Algorithm 1. Then a solution to the problem is given by* $T = k^*s$ *and* $(\mathbb{U}(0), \mathbb{U}(1), \ldots, \mathbb{U}(k^* - 1))$. □

The following example demonstrates Algorithm 1.

Example 2 Consider the network $\Sigma(3)$ in Fig. 3, where $X(0) =$ (PRJ, PRJ, OR, PRJ, OR, OR, OR). Assume that $s = 3$. Let us show how Algorithm 1 solves Problem 1, where f_{11}, f_{12}, f_{14}, and f_{21} are projection functions with respect to the first input and f_{13}, f_{22}, and f_{31} are logical OR.

In Step 1, we have $(x^*_{11}, x^*_{12}, x^*_{13}, x^*_{14}, x^*_{21}, x^*_{22}, x^*_{31}) =$ (PRJ, PRJ, OR, PRJ, PRJ, OR, PRJ). Then the algorithm generates the input sequence $\mathbb{U}(0) = (\hat{U}_{(2,1)}, \hat{U}_{(2,1)}, \hat{U}_{(2,1)})$ to flip the state of node (2, 1) in Step 3 at $k = 0$, where $\hat{U}_{(2,1)} = ((0, 0), (1, 0), (0, 0), (0, 0))$ as shown in Example 1. The result is shown in Fig. 6. At $k = 1$, we have the input sequence $\mathbb{U}(1) = (\hat{U}_{(3,1)}, \hat{U}_{(3,1)}, \hat{U}_{(3,1)})$ with $\hat{U}_{(3,1)} = ((1, 0, 0, 0), (1, 0, 0, 0))$ in Step 3, which flips the state of node (3, 1). As a result, we have the system in Fig. 7 with the output $Y(6) = v_{11}(6) \vee v_{13}(6) \vee w_{13}(6) \vee v_{14}(6)$. □

5 Conclusion

We have presented a learning method for a network of nodes each of which can implement classical conditioning. Based on the principle that the state of any node can be flipped while preserving the state of some other nodes, an algorithm has been derived.

As long as we know, the learning problem addressed in this paper has never been studied so far. This paper has presented the first solution to the problem, which proves

the feasibility of learning. On the other hand, the proposed algorithm may not be efficient in terms of required steps for learning. In fact, the proposed algorithm is based on the strategy that one gate is updated at a time, which results in taking a long time to achieve a desired state and might prevent us from applying it in an actual in vitro situation. Therefore, it is expected to develop a parallel algorithm, which updates multiple gates at the same time.

Moreover, the proposed algorithm is applicable to the case where the network is a tree structure and the state of each gate is either projection function or logical OR. It is also interesting to generalize our framework to handle a more general class of networks.

Acknowledgements This work was supported by Grant-in-Aid for Transformative Research Areas (A) 20H05969 from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

Data availability statement No datasets were generated or analysed during the current study.

Declarations

Conflict of Interest The author declares that he has no conflict of interest.

References

1. Murata, S., Toyota, T., Nomura, S.I.M., Nakakuki, T., Kuzuya, A.: Molecular cybernetics: challenges toward cellular chemical AI. Adv. Funct. Mater. **32**(37), 2201866 (2022)
2. Hart-Davis, A.: Pavlov's Dog: Groundbreaking Experiments in Psychology. Elwin Street Limited, London (2015)
3. Aggarwal, C.C.: Neural Networks and Deep Learning: A Textbook. Springer, Cham (2018)
4. Kauffman, S.: Homeostasis and differentiation in random genetic control networks. Nature **224**(5215), 177–178 (1969)
5. Sun, J., AlMomani, A.A.R., Bollt, E.: Data-driven learning of Boolean networks and functions by optimal causation entropy principle. Patterns **3**(11), 100631 (2022)
6. Robbin, J.W.: Mathematical Logic: A First Course. Dover Publications, New York (2006)
7. Hsiao, V., Hori, Y., Rothemund, P.W., Murray, R.M.: A population-based temporal logic gate for timing and recording chemical events. Mol. Syst. Biol. **12**(5), 869 (2016)
8. Lakin, M.R., Stefanovic, D.: Towards temporal logic computation using DNA strand displacement reactions. In 16th International Conference Unconventional Computation and Natural Computation, pp. 41–55 (2017)
9. O'Brien, J., Murugan, A.: Temporal pattern recognition through analog molecular computation. ACS Synth. Biol. **8**, 826–832 (2019)
10. Liu, C., Liu, Y., Zhu, E., Zhang, Q., Wei, X., Wang, B.: Cross-Inhibitor: a time-sensitive molecular circuit based on DNA strand displacement. Nucleic Acids Res. **48**(19), 10691–10701 (2020)
11. Yako, R., Ise, D., Komiya, K., Fujimoto, K., Kobayashi, S.: Monotone control of R systems. New Gener. Comput. **40**, 623–657 (2022)