Online Learning in a Chemical Perceptron

Peter Banda**
Portland State University

Christof Teuscher*,†
Portland State University

Matthew R. Lakin‡
University of New Mexico

Abstract Autonomous learning implemented purely by means of a synthetic chemical system has not been previously realized. Learning promotes reusability and minimizes the system design to simple input-output specification. In this article we introduce a chemical perceptron, the first full-featured implementation of a perceptron in an artificial (simulated) chemistry. A perceptron is the simplest system capable of learning, inspired by the functioning of a biological neuron. Our artificial chemistry is deterministic and discrete-time, and follows Michaelis-Menten kinetics. We present two models, the weight-loop perceptron and the weight-race perceptron, which represent two possible strategies for a chemical implementation of linear integration and threshold. Both chemical perceptrons can successfully identify all 14 linearly separable two-input logic functions and maintain high robustness against rate-constant perturbations. We suggest that DNA strand displacement could, in principle, provide an implementation substrate for our model, allowing the chemical perceptron to perform reusable, programmable, and adaptable wet biochemical computing.

Keywords
Perceptron, artificial chemistry, learning, adaptation, robustness, chemical computing

A version of this paper with color figures is available online at http://dx.doi.org/10.1162/artl_a_00105. Subscription required.

1 Introduction

Chemistry provides many beneficial features that contribute to information processing, such as inherent parallelism, massive interactivity, redundancy, and asynchronicity [2, 12, 36]. Biomolecular systems have successfully tackled several computing problems, including the traveling salesman [1], 3-SAT [6], maximal clique [41], chess [17], and tic-tac-toe [49]. However, attempts to build a programmable molecular automaton, that is, an automaton with more than one (hard-wired) purpose, either failed or had limited scope and no reusability [4, 11, 42].

Our approach is to achieve programmability of a chemical system by learning and adaptation. Adaptation is one of the key aspects maintaining the functional, homeostatic closure of *living* systems [5], which are carried out by chemical processes. It enables individual organisms to adjust their decision-making schemes in constantly changing environments. Learning has been a vibrant topic in the ALife community for over two decades. It has been realized by means of neural networks [21, 44], various forms of evolutionary algorithms [38, 39], and reinforcement learning [50], in which agents learn from the consequences of their actions through rewards. The applications of learning include pathfinding problems [30], multi-agent systems [34], and robotics [8].

* Contact author.
** Department of Computer Science, Portland State University, Portland, OR 97207. E-mail: banda@pdx.edu
† Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207. E-mail: teuscher@pdx.edu
‡ Department of Computer Science, University of New Mexico, Albuquerque, NM 87131. E-mail: mlakin@cs.unm.edu

The idea of neural network computation in chemical systems is not new. Several theoretical or experimental DNA-based models [7, 9, 24, 25, 32, 36] have been proposed. Most recently, Qian et al. [52] demonstrated an experimental implementation of linear threshold circuits with DNA-strand-displacement seesaw gates and used these to construct a Hopfield network. The research in this area has mainly been limited to constructing logic gates and assembling them into circuits to compute custom Boolean functions. More important, the existing work does not dwell on the autonomous learning aspects of chemical neural networks. Typically [32, 52] the learning was performed by an external system that computes the weights for a formal neural network, before converting these to molecular concentrations to serve as parameters for the chemical implementation.

Spiking neural P systems [27, 28] are related types of systems that draw inspiration from neural network theory and incorporate membrane computing with a model of spiking neurons. Each neuron is wrapped in a membrane, where interneuron (intermembrane) communication is carried by the electrical impulses, called spikes. Some attempts were made to introduce learning into neural P systems [20]; there, however, similarly to DNA-strand implementations, learning has not been autonomous either. Also, P systems do not model reaction rates, and they have a different, more grammarlike update algorithm (kinetics) and manipulate discrete symbols.

Here, we model a two-input perceptron, a simple learning unit, as an artificial chemistry [13], where both linear integration and learning are implemented internally. To the best of our knowledge, it is the first such model. We restrict the interactions with the perceptron solely to injections of training instances consisting of two inputs and the desired output, and measurements of the concentrations of the output species. Learning requires a system to work continuously; therefore, a cleanup or a reset to a steady state is necessary. Again, we do not assume any manual reset. We show that the chemical perceptron can learn a logic function perfectly, and is also robust to perturbations of rate constants.

We simulate a general (artificial) chemistry based on Michaelis-Menten kinetics without any assumption about the molecular structure of the chemical species. Since the behavior of species is fully determined by reactions and rates, our model imposes basic constraints on how real molecular species should interact in order to adapt. This abstraction allows us to better understand the underlying principles, as well as the inherent challenges of adaptable artificial chemistries. After a conversion of Michaelis-Menten to mass-action kinetics, DNA strand displacement [46, 53] can potentially serve as a biochemical implementation of our chemical perceptron.

The contributions of this work are as follows.

1. Our system is the first full-featured implementation of online learning in simulated artificial chemistry (Section 2), called the *chemical perceptron* (Section 4). Learning, as well as linear integration of weights and inputs, is handled internally.
2. We present a systematic method that maps the variables of a formal two-input perceptron (Section 3) to species of the chemical perceptron (Sections 2.1, and 4.1).
3. We implement two variants of a chemical perceptron, the weight-loop perceptron (Section 4.3) and the weight-race perceptron (Section 4.4), to demonstrate two qualitatively different approaches to linear integration, threshold comparison, and output production.
4. The chemical perceptron is reusable, since it recovers its internal ready state after each processing (Sections 4.2, 4.3, and 4.4).
5. The chemical perceptron learns perfectly all 14 linearly separable logic functions after 200 learning iterations (Section 7.1).
6. The chemical perceptron is robust to perturbations of rate constants (Section 7.2). This property helps to substantially alleviate reaction-timing restrictions for real chemical implementations.
7. We compare the weight-loop and weight-race perceptrons (Section 7.3) and discuss their potential biochemical implementation (Section 8).

2 Artificial Chemistry

Artificial chemistry (AC) is the standard framework for representing and simulating chemistry. An AC [13] consists of a set of molecular species or substances, S, and a set of reactions, R. Each reaction $r \in R$ is an ordered pair $X \rightarrow Y$, where both X and Y are multisets of species. Species from A are called reactants, and species from B are called products. Each reaction in our model is either a conversion ($A \rightarrow B$ or $A + B \rightarrow C + D$), an annihilation ($A + B \rightarrow \lambda$), or a decay ($A \rightarrow \lambda$), where λ represents no species.

There are many types of ACs, which can be simulated with various techniques. In this article we employ macroscopic deterministic simulation, where species interact on the basis of their reactions with associated rate laws [15, 16, 23]. Each reaction has a rate, which defines the strength of the reaction's contribution to the production or consumption of particular species over time. An essential property of macro-chemistry is the absence of space. In a well-stirred tank, the probability that a molecule is involved in a reaction does not depend on its position, but on its type. Consequently, a multiset of species, where each species is characterized only by a concentration measured in moles per liter (M), describes the state of the system. For instance, a state of the AC with the species set $S = \{A0, A1, B\}$ can be $[A0] = 2$ M, $[A1] = 2$ M, and $[B] = 10$ M.

Our AC follows the mass conservation principle, the mass-action law, Michaelis-Menten catalysis, and linear uncompetitive inhibition. The law of mass conservation states that in a closed system the matter consumed and produced by each reaction is the same. The mass-action law [3, 16] says that the rate of a reaction is proportional to the product of the concentrations of the reactants. Hence the reaction rate, the *speed* of the reaction application, is assumed to be linearly dependent on the concentration of reactants. For an irreversible generic reaction $aA + bB \rightarrow C$, the rate is given by

$$r = \frac{d[C]}{dt} = -\frac{1}{a}\frac{d[A]}{dt} = -\frac{1}{b}\frac{d[B]}{dt} = k[A]^a[B]^b, \tag{1}$$

where $k \in \mathbb{R}^+$ is a reaction rate constant, a, b are stoichiometric constants, and $[A]$, $[B]$ are the concentrations of molecular species A, B respectively.

Besides being a reactant or a product, species can take on two other roles in a reaction. A *catalyst* is a substance that increases the rate of a reaction without itself being altered. To incorporate catalysts (or enzymes) in reaction rates, we follow Michaelis-Menten enzyme kinetics [10, 35, 40]. Let $E + S \rightleftharpoons ES \rightarrow E + P$ be a catalytic reaction, where E is a catalyst, S is a substrate, P is a product, and ES is an intermediate enzyme-substrate binding species. Michaelis-Menten kinetics assumes that the substrate S is in instantaneous equilibrium with the enzyme-substrate complex ES. This assumption, called the quasi-steady-state approximation, holds if the first reaction $E + S \rightarrow ES$ is substantially faster than the second one $ES \rightarrow E + P$. The overall reaction rate for the P production is

$$r = \frac{d[P]}{dt} = \frac{k_{\text{cat}}[E][S]}{K_m + [S]}, \tag{2}$$

where $k_{\text{cat}}, K_m \in \mathbb{R}^+$ are rate constants. For example, let $R0$ be a reaction $A1 + B \rightarrow A0$, and $R1$ be a reaction $B \rightarrow A1$ catalyzed by $A0$ using the species set $S = \{A0, A1, B\}$. Then, the rate of $R0$ is expressed by the mass-action law as $k[A1][B]$, and the rate of $R1$ based on Michaelis-Menten kinetics as $k_0[A0][B]/(k_1 + [B])$, where k, k_0, and k_1 are rate constants. Michaelis-Menten kinetics can be also expanded to the multi-substrate case [33]. Note that an alternative to Michaelis-Menten kinetics is to use mass-action kinetics for two partial (associative and disassociative) reactions.

An *inhibitor* is a substance that retards the rate of a reaction without itself being consumed. There are several types of inhibition; our model is restricted to the simplest one, known as linear uncompetitive inhibition [33]. The reaction rate of $I + S \rightarrow I + P$, where I is an inhibitor and k and K_i are rate constants, is

$$r = \frac{d[P]}{dt} = -\frac{d[S]}{dt} = \frac{k[S]}{1 + K_i[I]}. \tag{3}$$

By applying rate laws over all reactions, we obtain the change of concentration of molecular species as described by a system of ordinary differential equations (ODEs). Since it is, in general, impossible to find an analytical solution of such a system explicitly, we employ numerical integration of ODEs, which delivers an approximate solution [46]. Our AC numerically integrates concentration ODEs by the simple one-step Euler method [26, 48].

Deterministic simulation of rate laws is fast and provides a good approximation of real chemistry if the number of molecules in the reservoir is sufficiently high. Otherwise, a microchemistry based on stochastic collisions of individual molecules [18, 19, 29, 51] produces results with higher precision, but comes at a higher simulation cost.

2.1 Representation of Variables

An AC can represent a variable by one or several substances. In our chemical model we need to encode variables of two types: `Boolean` with values 0 and 1, and `Real` with values from $\mathbb{R}$. We transform variables to species in systematic fashion as follows.

A `Boolean` variable is represented by enumeration; one variable s requires two species, S^0 and S^1, which are mutually exclusive. If the concentration of S^0 is nonzero, then $s = 0$; analogously S^1 nonzero implies $s = 1$. A positive value of a `Real` variable directly corresponds to the concentration of a species. Since a concentration is never negative, a `Real` variable s needs to be represented by positive and negative species $S^{\oplus}$ and $S^{\ominus}$. If both variants S^0 and S^1 for `Boolean`, or $S^{\oplus}$ and $S^{\ominus}$ for `Real`, are simultaneously present in the reservoir, they annihilate very rapidly.

Further, the zero concentration of species cannot represent a strict zero value. The problem is that intentional *nothing* cannot be distinguished from the state in which a system is still working and has not produced an output yet, or is still waiting for the input substances to enter the system.

2.2 System Input and Output

The concept of AC *actions* or *action series* is an extension of the input configuration—species concentrations can be modified at times other than t_0. An AC action emulates a step in the execution of an experimental protocol, where at a certain time the person performing the chemical experiment mechanically injects or removes substances into or out from a tank. An action is modeled by instantaneously changing the concentration of a species. For iterative processes, such as learning, it is useful to define a repetitive AC action series, in which a sequence of actions repeat in a loop at predefined time intervals.

An AC *translation*, also performed from outside the system, is used to interpret concentrations of output species as the results of the chemical computation over a specified time interval. Translations operate on concentration ranges; hence they are defined by aggregate functions, such as max, min, and $\sum$. Similarly to AC action series, AC translation series can be repetitive.

Figure 1 presents a concentration plot of species $S = \{A0, A1, B\}$ driven by reactions $R0$ and $R1$ as introduced in Section 2, using the rate constants $k = 0.00325$, $k_0 = 0.025$, and $k_1 = 0.5$. AC actions occur at times t_0, t_{100}, and t_{200} as described in the caption. By applying the AC translation

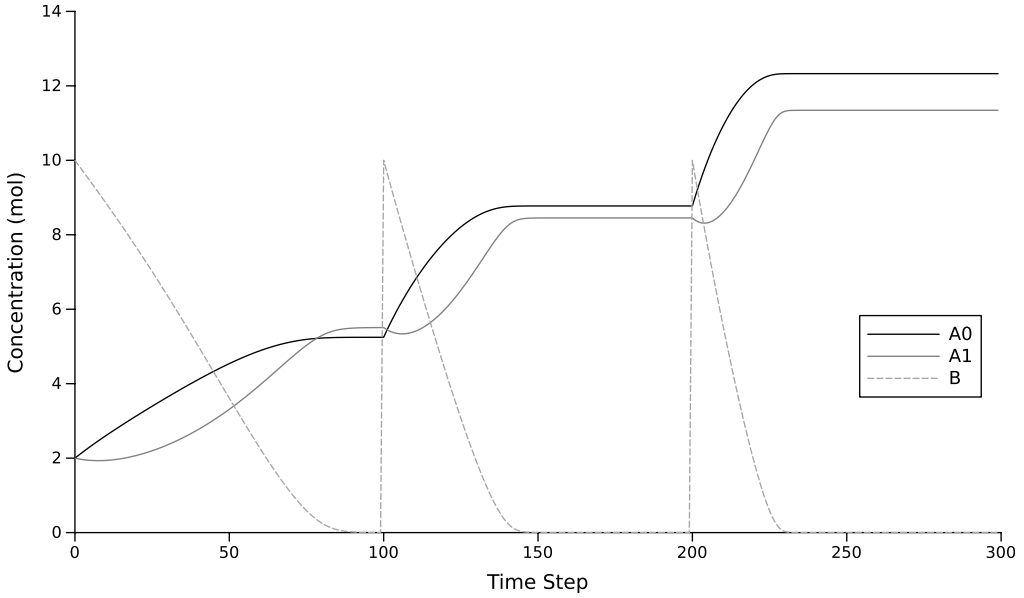

Figure 1. Concentration plot of an artificial chemistry with species $S = \{A0, A1, B\}$ and reactions $R = \{R0 : A1 + B \rightarrow A0, R1 : B \rightarrow A1$ with a catalyst $A0\}$. AC actions are provided at time step t_0: $[A0] = [A1] = 2$ M, and $[B] = 10$ M, t_{100}: $[B] = 10$ M, and t_{200}: $[B] = 10$ M. The AC translation max($[A1]$) > max($[A0]$) interprets the output sequence as 1, 0, 0 (from left to right).

defined by max($[A1]$) > max($[A0]$) on the intervals $t_0 - t_{99}$, $t_{100} - t_{199}$, and $t_{200} - t_{299}$, we translate the output to the sequence 1, 0, 0.

3 The Formal Perceptron

Artificial neural networks [44] are inspired by the coarse-grained behavior of biological neurons in the brain. The perceptron is an early type of artificial neural network and one of the simplest systems capable of learning [45].

A perceptron is a single neuron that processes a vector of input signals $\mathbf{x} = (x_1, \ldots, x_n)$, $x_i \in \mathbb{R}$, and produces one output y based on the setting of its weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ as shown in Figure 2. More precisely, a perceptron first calculates the linear integration (the dot product) of weights $\mathbf{w}$ and

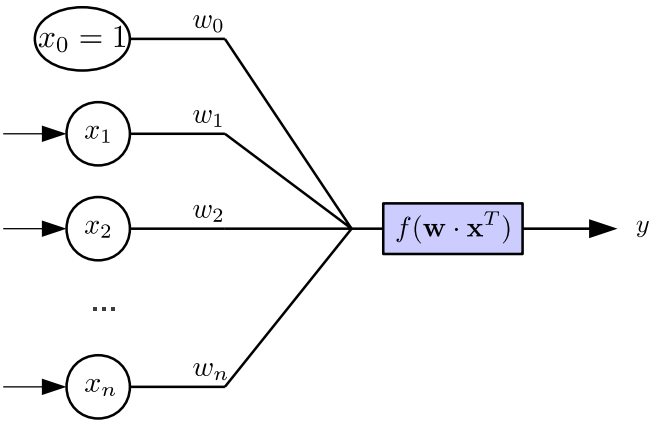

Figure 2. Model of a perceptron. An activation function f processes the dot product of weights and inputs $\mathbf{w} \cdot \mathbf{x}^T$, producing output y.

inputs $\mathbf{x}$ as $z = \sum_{i=0}^{n} w_i \cdot x_i$, and then passes the result z to an activation function $f : \mathbb{R} \to [0, 1]$ or $f : \mathbb{R} \to [-1, 1]$, which produces the final output y. Note that the weight w_0, called the bias or offset, always contributes to an output, since its associated input x_0 is a constant 1.

A perceptron can classify only linearly separable functions [37]—functions in which a straight line or, in the general case, a hyperplane can divide the inputs into two classes. By combining several perceptrons, we can construct a multilayer perceptron network, also known as a multilayer feed-forward network [21], that overcomes the linear separability problem and in fact becomes a universal approximator.

3.1 Learning

Perceptron learning [44] is a type of supervised Hebbian learning [22] where a training data set $T = \{(\mathbf{x_1}, d_1), \ldots, (\mathbf{x_m}, d_m)\}$ consisting of input-output pairs characterizes the target behavior of the system. During each step of the learning process, a perceptron absorbs one training sample $(\mathbf{x}, d)$. If there is a discrepancy between the actual output y and the desired output d, the error is fed back to the perceptron and triggers an adaptation of the participating weights. The adaptation of the weight w_i for the training sample $(\mathbf{x}, d)$ at time t is defined as $w_i(t + 1) = w_i(t) + \alpha(d - y(t))x_i$, where the learning rate $\alpha \in (0, 1]$ represents the adaptation strength.

If an error is detected, that is, if $|d - y| > 0$, the weight w_i shifts toward the desired output if its input signal x_i is nonzero. Conversely, if an input $x_i = 0$, the weight w_i is not involved in the global output y and therefore stays unaltered. Initially, the weights are set to small random values. The process of weight adaptation continues until the cumulative error of several consecutive training samples drops below the error threshold, or alternatively a fixed number of iterations is reached.

3.2 Two-Input Binary Perceptron

In this article we model a specific type of perceptron: the two-input perceptron with binary inputs x_1 and x_2, and real-value weights w_0, w_1, and w_2. The activation function $f =$ sgn (hard delimiter) outputs one if the dot product $z = w_0 + w_1x_1 + x_2w_2$ is positive, and zero otherwise. Therefore, the linear integration part is reduced to four cases as presented in Table 1a.

Table 1. (a) The relation of an input pair x_1 and x_2 to output y in the two-input binary perceptron. (b) Overview of the modeling capabilities of the two-input binary perceptron restricted to positive or negative values of weights w_0, w_1, w_2.

(a)			(b)																	
x_1	x_2	y	Weights			Binary functions														
0	0	$w_0 > 0$																		
1	0	$w_0 + w_1 > 0$	w_0	w_1	w_2	CONST0	NOR	NCIMPL	NOT X1	NIMPL	NOT X2	XOR	NAND	AND	NXOR	PROJ X2	IMPL	PROJ X1	CIMPL	OR	CONST1
0	1	$w_0 + w_2 > 0$	−	−	−	x															
1	1	$w_0 + w_1 + w_2 > 0$	−	−	+	x				x								x			
			−	+	−	x		x								x					
			−	+	+	x								x		x		x		x	
			+	−	−		x		x		x		x								x
			+	−	+						x								x		x
			+	+	−				x								x				x
			+	+	+																x

Now, we investigate whether all weights must support both positive and negative values. Assume w_0 is negative and w_1 and w_2 are positive. Then, for instance, the weights $w_0 = -10$, $w_1 = 7$, $w_2 = 9$ model the AND function, and the weights $w_0 = -10$, $w_1 = 12$, $w_2 = 13$ the OR function. However, no combination of negative w_0 and positive w_1 and w_2 weight values can represent (e.g.) the NAND function. Table 1b summarizes the limitations of all sign-weight combinations for a representation of logic functions. It shows that each of the weights w_0, w_1, and w_2 must support both positive and negative values to implement a perceptron that can encompass all 14 linearly separable binary functions, that is, all binary functions except XOR and NXOR.

4 The Chemical Perceptrons

In this section we describe the implementation of the two-input perceptron by means of an artificial chemistry. We want to emphasize that there are many ways to approach this problem. Here we present two models—the weight-loop perceptron and the weight-race perceptron—which represent two fundamental, substantially different techniques for a calculation of the weight sum and the zero threshold. Before we describe the weight-loop and weight-race perceptrons in detail, first we need to formalize the features shared by both models.

4.1 Species

The representation of the formal perceptron's variables by chemical species follows the scheme presented in Section 2.1. Each variable of type Boolean needs 0-value and 1-value species, and each variable of type Real splits into $\oplus$ and $\ominus$ species variants. Table 2a presents the core species of a chemical perceptron, along with their mappings to the variables of the formal perceptron described in Section 3.

The inputs x_1 and x_2 of type Boolean trigger different processing paths in the perceptron, based on their values. Therefore, each input-variable value pair requires its own species: X_1^0 and X_1^1 to represent x_1, and X_2^0 and X_2^1 to represent x_2. Note that if a single species encoded an input variable, we would need to differentiate values 0 and 1 based on low versus high concentration, which would lead to a more complicated design. Similarly, variables y and d are also Booleans, so the same representation applies: Y^0, Y^1 and D^0, D^1. Weights of the two-input binary perceptron need to support both positive and negative values (Table 1b), so each weight w_i splits into two distinct positive and negative variants, species $W_i^{\oplus}$ and $W_i^{\ominus}$, where $i \in \{0, 1, 2\}$. For simplification, we will use the name of a group or a subgroup to refer to all associated species as defined in Table 2a. For example, the species subgroup X_1 includes both X_1^0 and X_1^1, and the group X includes species X_1^0, X_1^1, X_2^0, and X_2^1.

4.2 Binary-Function and Learning Modes

The chemical perceptron can function in two modes: *binary-function mode* and *learning mode*. In the binary-function mode, the perceptron basically acts like a logic gate; it takes two inputs X_1 and X_2, and produces an output Y. The second learning mode is built on top of the output production, so again inputs must be present. The learning is triggered by the desired-output molecules D. Once output Y is produced, it is compared against D, and if they differ (i.e., Y^0 is produced but D^1 is expected, or vice versa), positive- or negative-weight molecules are created and added to existing ones, to modify the weights.

After each learning iteration, the perceptron needs to recover to its steady state. Hence, *transient* species not consumed during the chemical computation must be removed by some cleanup reaction such as decay, because there is no external cleanup. Only the weight species form the persistent state of the chemical perceptron; hence, all other species are considered transient. Since decay and annihilation reactions would break the mass-conservation principle, we assume that instead of nothing (λ), an inert by-product is produced.

Table 2. (a) The core species of the chemical perceptron with the mapping to the variables of a perceptron. Each variable is either of type `Boolean` with the domain {0, 1}, or `Real` with the domain $\mathbb{R}$. The nonzero concentration of species represents a value of the variable on a domain subset as specified by the domain restriction column. The name of a group or subgroup is used to refer to the associated species. (b) The extra species of the weight-loop perceptron, which do not map to the variables of a formal perceptron: the processed weights $\overline{W}$, and the fuel E. Again, the name of a group or subgroup is used to refer to the associated species.

(a)						(b)		
Group	Subgroup	Variable	Domain	Species	Domain restriction	Group	Subgroup	Species
X	X_1	x_1	{0, 1}	X_1^0	$x_1 = 0$	$\overline{W}$	$\overline{W}_0$	$\overline{W}_0^{\oplus}$
				X_1^1	$x_1 = 1$			$\overline{W}_0^{\ominus}$
	X_2	x_2	{0, 1}	X_2^0	$x_2 = 0$		$\overline{W}_1$	$\overline{W}_1^{\oplus}$
				X_2^1	$x_2 = 1$			$\overline{W}_1^{\ominus}$
							$\overline{W}_2$	$\overline{W}_2^{\oplus}$
Y		y	{0, 1}	Y^0	$y = 0$			$\overline{W}_2^{\ominus}$
				Y^1	$y = 1$			
						E		E
D		d	{0, 1}	D^0	$d = 0$			
				D^1	$d = 1$			
W	W_0	w_0	$\mathbb{R}$	$W_0^{\oplus}$	$w_0 > 0$			
				$W_0^{\ominus}$	$w_0 < 0$			
	W_1	w_1	$\mathbb{R}$	$W_1^{\oplus}$	$w_1 > 0$			
				$W_1^{\ominus}$	$w_1 < 0$			
	W_2	w_2	$\mathbb{R}$	$W_2^{\oplus}$	$w_2 > 0$			
				$W_2^{\ominus}$	$w_2 < 0$			

Due to the multiplicity of input, output, desired-output, and weight species, reactions of the same type are collected into groups, which simplifies the reasoning as well as the simulation of the chemical perceptron. Reactions belonging to the same group share common structural characteristics, catalysts, and inhibitors, as well as rate constants.

In the following sections we present the species and reactions of the weight-loop and weight-race perceptrons. The actual setting of the reaction rate constants, which is optimized by the genetic algorithm, is discussed separately in Section 6.

4.3 Weight-Loop Perceptron

The weight-loop perceptron follows the formal perceptron definition from Section 3 quite rigorously. It consists of 21 species and 34 reactions. Apart from the core species defined in Section 4.1, the weight-loop perceptron requires the processed-weight species $\overline{W}_i^{\oplus}$ and $\overline{W}_i^{\ominus}$, $i \in \{0, 1, 2\}$, and fuel species E (Table 2b).

The weight-loop perceptron computes the weight sum directly by transforming weights W into output species Y. The problem is that the weights encode the state of the perceptron, so their concentration must be preserved. Therefore, apart from Y species, the perceptron must also create backup copies of the weights, $\overline{W}$. The perceptron can then restore its weights after the output production is over. A reaction $W_i \rightarrow \overline{W}_i + Y$ followed by $\overline{W}_i \rightarrow W_i$ would break the mass-conservation law, so the perceptron needs to consume a fuel, species E, which is provided to the system at constant concentration 1 M. From a functional perspective, the perceptron sequentially processes an input, produces an output, recovers weights, and finally performs a cleanup (Figure 3).

The perceptron starts working when inputs X_1 and X_2 are injected into the system. The perceptron processes the weight W_1 on input X_1^1, the weight W_2 on input X_2^1, and the weight W_0, in parallel, producing Y^0 and Y^1 molecules. Species X_1^1, formally encoding $x_1 = 1$, catalyzes $\oplus$ and $\ominus$ versions of reaction $W_1 + E \rightarrow \overline{W}_1 + Y$. Similarly, species X_2^1, which represents $x_2 = 1$, catalyzes $W_2 + E \rightarrow \overline{W}_2 + Y$. Since the weight W_0 always contributes to the sum regardless of input, each of the possible inputs X_1^0, X_1^1, X_2^0, and X_2^1 catalyzes $W_0 + E \rightarrow \overline{W}_0 + Y$. In order to determine whether the total concentration of Ys is above or below the zero threshold, we let Y^0 annihilate with Y^1. If there are more Y^0 molecules at the end, the output is 0; otherwise 1. Weights could alternate between the normal version W and the processed version $\overline{W}$, each time consuming a fuel E and producing new Y molecules. To prevent a continuous cycling of the weights, the weight-loop perceptron must ensure that there is no input present before it rolls the weights back. That is, input species must decay, and the processed weights roll back only when substantial amounts of inputs are gone—that is, inputs act as inhibitors on $\overline{W} \rightarrow W$ reactions. The output molecules Y are removed from the system by a decay. Table 3a presents the full set of reactions with associated catalysts and inhibitors.

Only eight reactions are needed for the learning part (Table 3a, groups 9–11). During each learning step, the actual output Y is compared against the desired output D. If Y matches D (i.e., the substances Y^0 and D^0, or Y^1 and D^1, are simultaneously present in the system), the output is correct. In this case, no learning is needed and the weights remain unaltered. Otherwise, the desired-output species D transforms to $\oplus$ or $\ominus$ versions of the weight species W, but only for those participating in an output production for current inputs. Thus, an input and an output together catalyze the $D \rightarrow W$ reactions, so they are dependent (AND) catalysts (Table 3a, group 11).

For instance, if the perceptron produces the output Y^0 for the input species X_1^0 and X_2^1, but the desired output D^1 is injected, then reactions $D^1 \rightarrow W_2^{\oplus}$ and $D^1 \rightarrow W_0^{\oplus}$ are triggered, and weights $W_2^{\oplus}$

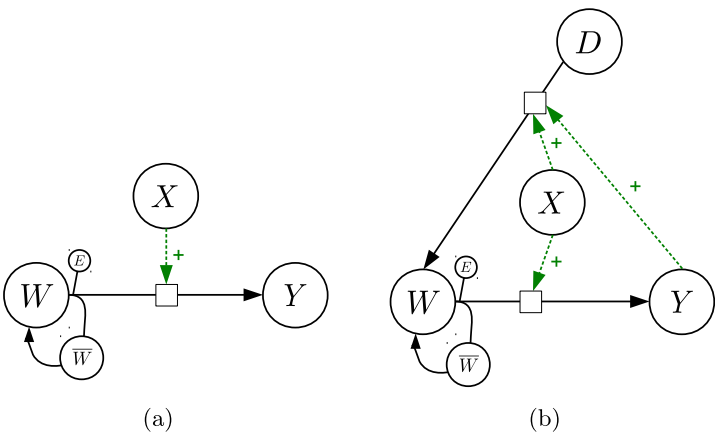

Figure 3. Qualitative diagram of the weight-loop perceptron's reactions. Each node represents a group of species, solid lines are reactions, squares are reaction rates, and dashed lines with a + sign are catalyses. (a) In the binary function mode, the input species X trigger (catalyze) a reaction $W + E \rightarrow \overline{W} + Y$, which consumes weight W and fuel E, and produces output Y and a backup copy of the weight, $\overline{W}$. After the output is produced, the weight-loop chemical perceptron recovers the weights by reactions $\overline{W} \rightarrow W$. (b) In the learning mode, the desired-output species D^0 or D^1 transforms to weights W, if the provided variant does not match the actual output, Y^0 or Y^1.

Table 3. (a) Reactions of the weight-loop perceptron with associated catalysts and inhibitors. (b) Reactions of the weight-race perceptron with associated catalysts. Reactions are divided into the groups according to the common functional characteristics. Reaction types are C (conversion), A (annihilation), D (decay).

(a)

Group	Type	Reaction	Catalysts	Inhibs
1	C	$W_0^{\oplus} + E \to \overline{W}_0^{\oplus} + Y^1$	X	
		$W_0^{\ominus} + E \to \overline{W}_0^{\ominus} + Y^0$	X	
2	C	$W_1^{\oplus} + E \to \overline{W}_1^{\oplus} + Y^1$	X_1^1	
		$W_1^{\ominus} + E \to \overline{W}_1^{\ominus} + Y^0$	X_1^1	
		$W_2^{\oplus} + E \to \overline{W}_2^{\oplus} + Y^1$	X_2^1	
		$W_2^{\ominus} + E \to \overline{W}_2^{\ominus} + Y^0$	X_2^1	
3	C	$W_1^{\oplus} \to \overline{W}_1^{\oplus}$	X_1^0	
		$W_1^{\ominus} \to \overline{W}_1^{\ominus}$	X_1^0	
		$W_2^{\oplus} \to \overline{W}_2^{\oplus}$	X_2^0	
		$W_2^{\ominus} \to \overline{W}_2^{\ominus}$	X_2^0	
4	C	$\overline{W}_0^{\oplus} \to W_0^{\oplus}$		X
		$\overline{W}_0^{\ominus} \to W_0^{\ominus}$		X
		$\overline{W}_1^{\oplus} \to W_1^{\oplus}$		X
		$\overline{W}_1^{\ominus} \to W_1^{\ominus}$		X
		$\overline{W}_2^{\oplus} \to W_2^{\oplus}$		X
		$\overline{W}_2^{\ominus} \to W_2^{\ominus}$		X
5	A	$W_0^{\oplus} + W_0^{\ominus} \to \lambda$		
		$W_1^{\oplus} + W_1^{\ominus} \to \lambda$		
		$W_2^{\oplus} + W_2^{\ominus} \to \lambda$		
6	A	$Y^0 + Y^1 \to \lambda$		

(b)

Group	Type	Reaction	Catalysts
1	C	$X_1^0 \to Y^1$	$W_0^{\oplus}$
		$X_1^0 \to Y^0$	$W_0^{\ominus}$
		$X_2^0 \to Y^1$	$W_0^{\oplus}$
		$X_2^0 \to Y^0$	$W_0^{\ominus}$
		$X_1^1 \to Y^1$	$W_0^{\oplus}$
		$X_1^1 \to Y^0$	$W_0^{\ominus}$
		$X_2^1 \to Y^1$	$W_0^{\oplus}$
		$X_2^1 \to Y^0$	$W_0^{\ominus}$
2	D	$X_1^0 \to \lambda$	$W_0^{\oplus}$
		$X_1^0 \to \lambda$	$W_0^{\ominus}$
		$X_2^0 \to \lambda$	$W_0^{\oplus}$
		$X_2^0 \to \lambda$	$W_0^{\ominus}$
3	C	$X_1^1 \to Y^1$	$W_1^{\oplus}$
		$X_1^1 \to Y^0$	$W_1^{\ominus}$
		$X_2^1 \to Y^1$	$W_2^{\oplus}$
		$X_2^1 \to Y^0$	$W_2^{\ominus}$
4	C	$W_0^{\oplus} + W_0^{\ominus} \to \lambda$	
		$W_1^{\oplus} + W_1^{\ominus} \to \lambda$	
		$W_2^{\oplus} + W_2^{\ominus} \to \lambda$	
5	A	$Y^0 + Y^1 \to \lambda$	
6	D	$Y^0 \to \lambda$	
		$Y^1 \to \lambda$	

Table 3. (*continued*).

(a)					(b)			
Group	Type	Reaction	Catalysts	Inhibs	Group	Type	Reaction	Catalysts
7	D	$X_1^0 \to \lambda$			7	D	$D^0 \to \lambda$	
		$X_1^1 \to \lambda$					$D^1 \to \lambda$	
		$X_2^0 \to \lambda$			8	C	$D^0 \to W_0^{\ominus}$	Y^1
		$X_2^1 \to \lambda$					$D^1 \to W_0^{\oplus}$	Y^0
8	D	$Y^0 \to \lambda$			9	C	$D^0 \to W_1^{\ominus}$	Y^1, X_1^1 (AND)
		$Y^1 \to \lambda$					$D^0 \to W_2^{\ominus}$	Y^1, X_2^1 (AND)
9	D	$D^0 \to \lambda$					$D^1 \to W_1^{\oplus}$	Y^0, X_1^1 (AND)
		$D^1 \to \lambda$					$D^1 \to W_2^{\oplus}$	Y^0, X_2^1 (AND)
10	C	$D^0 \to W_0^{\ominus}$	Y^1					
		$D^1 \to W_0^{\oplus}$	Y^0					
11	C	$D^0 \to W_1^{\ominus}$	Y^1, X_1^1 (AND)					
		$D^0 \to W_2^{\ominus}$	Y^1, X_2^1 (AND)					
		$D^1 \to W_1^{\oplus}$	Y^0, X_1^1 (AND)					
		$D^1 \to W_2^{\oplus}$	Y^0, X_2^1 (AND)					

and $W_0^{\oplus}$ are produced and added to (or annihilate with) existing weights. The strength of the adaptation (i.e., the learning rate α) is incorporated into the concentration of the desired output species D. For example, if 10 M of D^1 is injected into the system in the previous example, that 10 M is distributed between production of $W_0^{\oplus}$, and $W_2^{\oplus}$ (a small amount of D^1 actually disappears because of a decay reaction).

Since the system is open and weights W can switch reversibly to $\overline{W}$, consuming a fuel E provided from outside, an infinite loop might emerge, in which the concentrations of Y molecules increase without bound. The correct timing of phases is crucial for avoiding this problem.

4.4 Weight-Race Perceptron

The functioning of the weight-loop perceptron is based on rather conservatively designed phases working in a sequence. This approach works well, since there is almost a one-to-one relation between the routines of the formal perceptron and those of the chemical perceptron. Nevertheless, the idea of direct calculation of the weight sum and recovering the original state seems unnecessarily cumbersome for a chemical system.

The weight-race perceptron improves on the weight-loop model by switching the chemical roles of inputs and weights—that is, instead of having inputs catalyzing a transformation of weights to a weight sum, which determines an output, weights simply catalyze the input-to-output reactions as

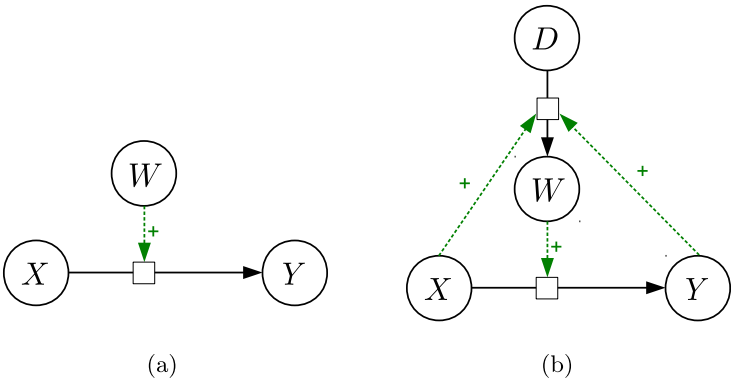

Figure 4. Qualitative diagram of the weight-race perceptron's reactions. Each node represents a group of species, solid lines are reactions, squares are reaction rates, and dashed lines with + sign are catalyses. (a) In the binary-function mode, the weights W catalyze (compete on) the input-to-output reactions $X \to Y$. (b) In the learning mode, the desired-output species D^0 or D^1 transforms to weights W if the provided variant does not match the actual output, Y^0 or Y^1.

presented in Figure 4. Thus, the perceptron does not compare weights directly, but lets them compete on input-to-output reactions as catalysts, so it basically implements a rate (derivation-based) comparison. For this to work, species Y^0 must annihilate with Y^1 quickly, racing must be simultaneous, and the rate functions must have similar shapes.

The weight-race chemical perceptron consists of 14 species and 30 reactions. Input species X_1 and X_2 are transformed directly to Y^0 or Y^1, depending on the signs of the currently present weights. Weight W_1 solely catalyzes reaction $X_1^1 \to Y^1$ for variant $W_1^{\oplus}$, and $X_1^1 \to Y^0$ for variant $W_1^{\ominus}$. Similarly, weight W_2 drives two reactions, in which X_2^1 is the reactant. Since weight W_0 is always active, it drives all possible input-to-output reactions. Weights catalyze the reactions concurrently, so the one with the highest concentration consumes the largest portion of an input and therefore has the highest contribution in an output. Analogously to the weight-loop model, an annihilation of Y^1 and Y^0 decides whether the concentrations of the Y's are above or below the zero threshold. The full set of reactions with associated catalysts is presented in Table 3b. The learning part is the same as in the weight-loop model.

If the weight-race perceptron is to treat all weights equally, it must ensure that the weight race is fair. Following the formal perceptron definition, the contribution of weights in the sum must be uniform, meaning there is no preference among weights. Apart from the concentrations of weights, the reaction rate constants determine the actual speed of the input consumption. Now, if all weights have the same sign, then it does not matter what rate constants are set, so let the qualitative state of the perceptron be $W_0^{\oplus}$, $W_1^{\ominus}$, and $W_2^{\ominus}$ (Figure 5).

For inputs X_1^0 and X_2^0, only the weight $W_0^{\oplus}$ is active, so there is no racing. The weight $W_0^{\oplus}$ competes with $W_1^{\ominus}$ and $W_2^{\ominus}$ for inputs X_1^1 and X_2^1; however, $W_0^{\oplus}$ is privileged, since it consumes

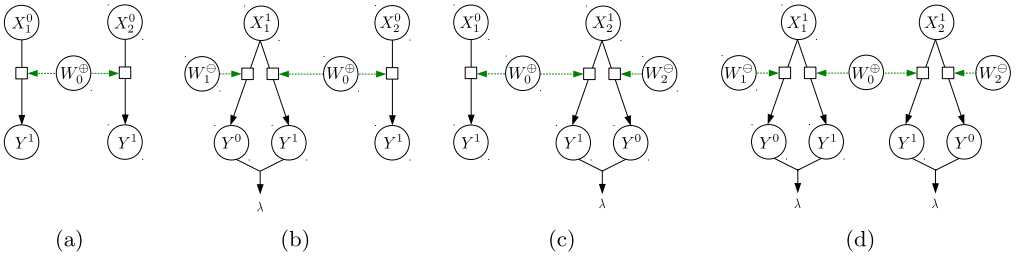

Figure 5. The input-output precessing of the weight-race perceptron for positive W_0 and negative W_1 and W_2 weights and for inputs: (a) (0, 0), (b) (1, 0), (c) (0, 1), (d) (1, 1).

X_1^1 and X_2^1 at the same time. Note that $W_1^{\ominus}$ consumes just X_1^1, and $W_2^{\ominus}$ just X_2^1. To avoid this problem we have to penalize $W_0^{\oplus}$ by setting the rate constants of the reactions $X_1^1 \rightarrow Y^1$ and $X_2^1 \rightarrow Y^1$, both catalyzed by $W_0^{\oplus}$, to δ, and those catalyzed by $W_1^{\ominus}$ and $W_2^{\ominus}$ to 2δ. As a result, the contribution preference of the weights is balanced.

A new problem emerges for inputs X_1^1 and X_2^0, where $W_0^{\oplus}$ drives two, but $W_1^{\ominus}$ just one reaction. Even if the X_1^1 reactions follow the two-to-one rate constant ratio, whatever constant is assigned to the reaction $X_2^0 \rightarrow Y^1$ catalyzed by $W_0^{\oplus}$ will result in an unfair advantage for $W_0^{\oplus}$, since eventually all X_2^0 molecules will change to Y^1. To balance the preference of $W_0^{\oplus}$ we need to introduce a decay of X_2^0, such that exactly two-thirds is taken away. In order to do that, $W_0^{\oplus}$ must catalyze not just $X_2^0 \rightarrow Y^1$ with the rate constant δ, but also the decay $X_2^0 \rightarrow \lambda$ with the rate constant 2δ. That is the reason why the reaction set of the weight-race perceptron contains a decay of the input species X_1^0 and X_2^0 (Table 3b, group 2).

The two-to-one ratio of rate constants must hold if the goal is to model the perceptron with no preference among weights. However, as we show in Section 6, if the weight-race perceptron does not obey this ratio and has a bias on a particular weight, it can still perform well.

Compared to the previous model, the weight-race perceptron is substantially simpler. It is minimal in number of species (only the core set is used), and it contains just 30 reactions, without any inhibition. Unlike the weight-loop perceptron, the system does not need any externally supplied fuel species. In fact, the input species adopt this role, so they are essentially an information and energy source.

5 Specifying Execution Settings

We have introduced the chemical perceptron models structurally as collections of species and reactions with catalysts and inhibitors. Now, we shall specify the setting of executions (or chemical experiments) in terms of the action series and the translation series (Section 2.2).

5.1 Binary-Function Modeling

Since the input processing and the output production take some time, we cannot inject input species X_1 and X_2 immediately after the previous pair, but we have to wait for a certain number of simulation steps, $S = 5{,}000$. In the binary-function mode, the first action at time t_0, handling an initialization sets the weights W according to the target logic function. Then every S steps we execute one of the four actions (one per input pattern) for the weight-loop perceptron:

- $[X_1^0] = 1$ M, $[X_2^0] = 1$ M,
- $[X_1^1] = 1$ M, $[X_2^0] = 1$ M,
- $[X_1^0] = 1$ M, $[X_2^1] = 1$ M,
- $[X_1^1] = 1$ M, $[X_2^1] = 1$ M.

In the weight-loop perceptron, inputs serve as catalysts of the weight-to-output reactions, which consume fuel species E. Since the weight-race perceptron directly transforms input species to output species, the concentration of inputs must be higher:

- $[X_1^0] = 2$ M, $[X_2^0] = 2$ M,
- $[X_1^1] = 2$ M, $[X_2^0] = 2$ M,
- $[X_1^0] = 2$ M, $[X_2^1] = 2$ M,
- $[X_1^1] = 2$ M, $[X_2^1] = 2$ M.

Figure 6 presents simulations of the weight-loop perceptron and the weight-race perceptron, each computing the NAND function on four consecutive inputs.

5.2 Learning

In learning mode, the initial concentrations of weights are generated randomly in the interval 2–10 M, with equal probability of selecting positive and negative variants. A learning rate α, which is constant throughout the whole training, is incorporated into the concentration of the desired output D.

The original definition of perceptron learning (Section 3) adjusts weight w_i by $\Delta w_i = \alpha(d - y)x_i$ for a given output y and desired output d. Assuming $d \neq y$, each weight participating in the output production increments by $\Delta w = \alpha(d - y)$. Since the sign of the weight sum fully determines output, weight adaptation is stronger for inputs with the higher number of ones. For instance, the weight sum is adjusted by Δw for input (0, 0), but by $3\Delta w$ for input (1, 1), as shown in Table 4a.

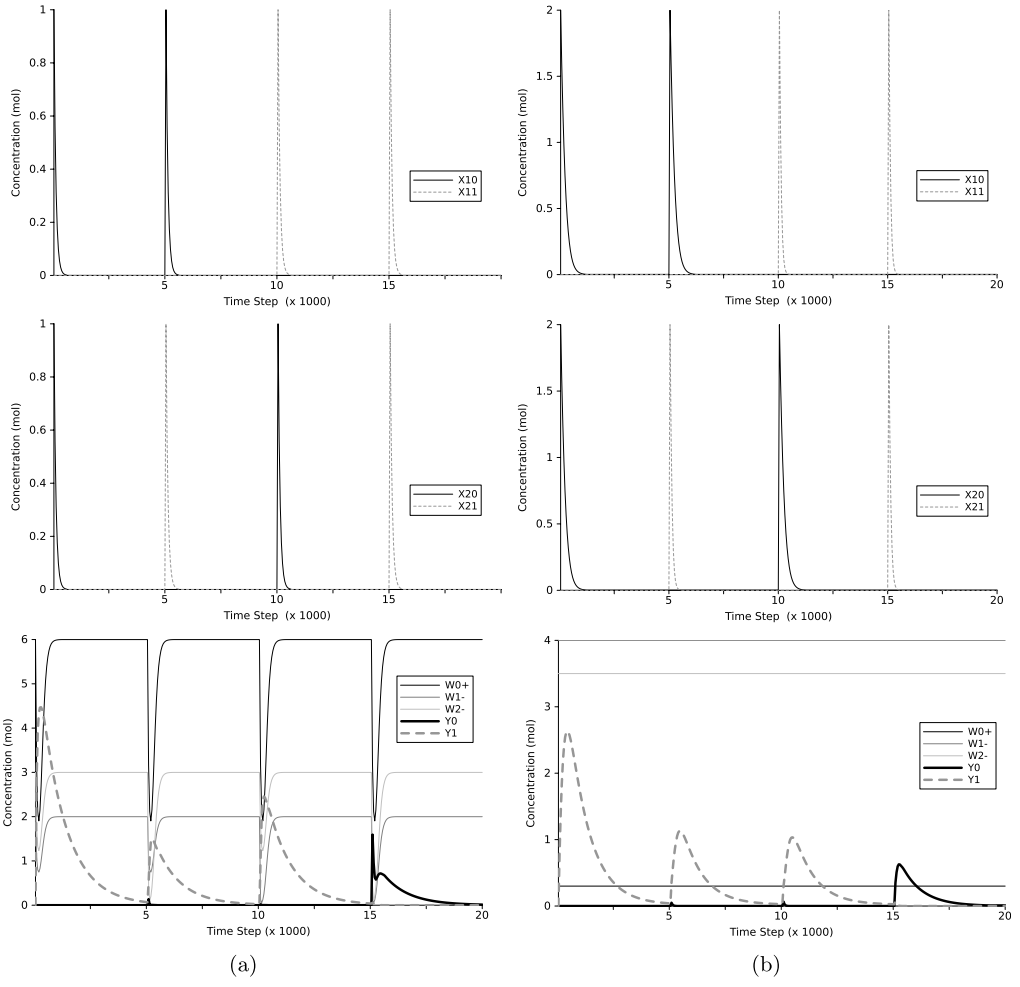

Figure 6. Simulation of the NAND function on four different combinations of the input species by (a) the weight-loop perceptron with initial weight concentrations $[W_0^{\oplus}] = 6$ M, $[W_1^{\ominus}] = 2$ M, and $[W_2^{\ominus}] = 3$ M, and (b) the weight-race perceptron with initial weight concentrations $[W_0^{\oplus}] = 0.3$ M, $[W_1^{\ominus}] = 4$ M, and $[W_2^{\ominus}] = 3.5$ M. From top to bottom: concentrations of input species X_1^0 and X_1^1, concentrations of input species X_2^0 and X_2^1, and concentrations of output Y and weight species W. By applying the translation that compares the maximal concentrations of Y^1 and Y^0 in the four intervals 5,000 steps long, we obtain the NAND output sequence 1, 1, 1, 0.

Table 4. The adaptation of a weight sum during learning by a two-input binary perceptron for four inputs: (a) a uniform adaptation of individual weights, (b) a uniform adaptation of the weight sum.

(a)			(b)		
x_1	x_2	Adapted weight sum	x_1	x_2	Adapted weight sum
0	0	$w_0 + \Delta w$	0	0	$w_0 + \Delta w$
1	0	$w_0 + w_1 + 2\Delta w$	1	0	$w_0 + w_1 + \Delta w$
0	1	$w_0 + w_2 + 2\Delta w$	0	1	$w_0 + w_2 + \Delta w$
1	1	$w_0 + w_1 + w_2 + 3\Delta w$	1	1	$w_0 + w_1 + w_2 + \Delta w$

In the chemical perceptron, the concentration of the desired output is divided between weights; hence the weight adaptation of the original perceptron would correspond to the concentration profile of the desired output for four consecutive inputs: $[D] = \alpha$, $[D] = 2\alpha$, $[D] = 2\alpha$, $[D] = 3\alpha$. Our simulations proved that this unfairness results in a split of performance for function pairs, such as `CONST0`, `CONST1` or `AND`, `NAND`. Since these functions are the inverse of each other, the chemical perceptron should learn them with the same success.

Essentially, the uniform adaptation of individual weights causes a bias in the weight adaptation. To avoid that, we do not adapt individual weights, but the whole weight sum uniformly for all inputs, as presented in Table 4b. More specifically, the chemical perceptron divides Δw, the concentration of the desired output D among weights, so the whole weight sum is adjusted by Δw. Note that a small amount of the desired output disappears because of a decay.

The concentration of the desired-output species D is constant for all inputs in the chemical perceptron. By experiments we determined that the optimal concentration of D is 2 M. If the concentration was too high, the weights would oscillate and would not converge on a stable solution. Conversely, a low concentration of D prolongs the learning process and does not provide enough pressure to drive weights out of the zero region if their concentrations are very low.

As opposed to our chemical perceptron, the biased adaptation in the original formal perceptron does not cause substantial problems, because the weight sum is further processed by an activation function and the learning rate α decreases over time. As a result, small differences in the weight adaptation become unimportant.

Let $f\colon \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$ be a target two-input logic function, which we wish to teach to the perceptron. By $D^{f(x_1, x_2)}$ we denote species D^0 if $f(x_1, x_2) = 0$ or D^1 if $f(x_1, x_2) = 1$. The training set of the weight-loop perceptron consists of four actions:

- $[X_1^0] = 1$ M, $[X_2^0] = 1$ M, $[D^{f(0,0)}] = 2$ M,
- $[X_1^1] = 1$ M, $[X_2^0] = 1$ M, $[D^{f(1,0)}] = 2$ M,
- $[X_1^0] = 1$ M, $[X_2^1] = 1$ M, $[D^{f(0,1)}] = 2$ M,
- $[X_1^1] = 1$ M, $[X_2^1] = 1$ M, $[D^{f(1,1)}] = 2$ M.

Similarly, the training set of the weight-race perceptron is:

- $[X_1^0] = 2$ M, $[X_2^0] = 2$ M, $[D^{f(0,0)}] = 2$ M,
- $[X_1^1] = 2$ M, $[X_2^0] = 2$ M, $[D^{f(1,0)}] = 2$ M,
- $[X_1^0] = 2$ M, $[X_2^1] = 2$ M, $[D^{f(0,1)}] = 2$ M,
- $[X_1^1] = 2$ M, $[X_2^1] = 2$ M, $[D^{f(1,1)}] = 2$ M.

Learning consists of a series of actions, each randomly chosen from the training set and performed every S steps. The total number of actions, L, per action series is $L_f = 120$ for the fitness evaluation (Section 6), and $L_p = 200$ for the learning performance and robustness analysis (Sections 7.1 and 7.2), so the perceptron runs for either $S \times L_f = 6 \times 10^5$ or $S \times L_p = 10^6$ time steps.

If we injected inputs together with the desired output at the same time, the adaptation of the weights would start immediately, changing the actual output, so the actual output would differ from the one we would obtain by providing only input species. In the extreme case, the chemical perceptron could just copy the desired output to the actual output by having very low concentrations of weights. To prevent this, we inject an input, wait a certain number of steps, measure the output, and then provide the desired output. Note that a reaction $D \rightarrow W$ requires both catalysts, the input species X and the output species Y, to have a sufficient concentration at the moment of weight adaptation. Therefore, we must allow enough time for the output production, but we cannot postpone the injection of D for too long; if we did, the chemical perceptron would process or decay all input species. We found experimentally that this delay can be fairly short. More precisely, in our learning simulations we inject the desired output 100 steps after the input species.

Now, the only question is how to interpret the output, or in other words, how the concentrations of species Y^0 and Y^1 translate to the value of the variable y from the formal definition. Since y is a `Boolean` variable, the translation compares the concentrations of value-0 species and value-1 species just before a desired output is injected. Hence, the value of the variable y is defined as $[Y^1] > [Y^0]$ at relative time step 99.

6 Choosing Rate Constants

Since the number of rate constants of the chemical perceptrons is very large, it would be difficult and time-consuming to set them manually in a trial-and-error fashion or by exhaustive search. We therefore optimize the setting of the rate constants by employing a standard genetic algorithm (GA) [14, 43].

Because the reactions from the same group are structurally similar and in fact they share the same rate constants, the total number of representative rate constants constituting a possible solution, known as a *chromosome*, can be reduced substantially: from 68 to 21 for the weight-loop and from 52 to 14 for the weight-race perceptron. Figure 7 shows the chromosome structure for both perceptrons.

The fitness of a chromosome is evaluated as the performance of a chemical perceptron with the given rate constants (from the chromosome) on the binary-function learning task. Whether a chemical perceptron has learned a given function depends primarily on its final state; hence, the fitness embraces the performance for only the last steps. More precisely, the fitness is the Hamming distance between the last $P = 20$ translated actual outputs $y_{L-P},\ldots, y_L$ and the desired outputs $d_{L-P},\ldots, d_L$. Because each learning action series starts with a random setting of weight concentrations, we calculate the fitness over 30 runs for each of the 14 functions.

Due to the nondeterministic nature of the action series, we find the fitness score more comparable across the current population if the chromosomes obtain the same problem instances by preevaluating action series. The GA combines elite selection with one point crossover and per-element (rate constant) mutation. Since only certain values of rate constants are plausible, we restrict their value ranges for the mutation and the generation of the initial population as shown in Table 5. The

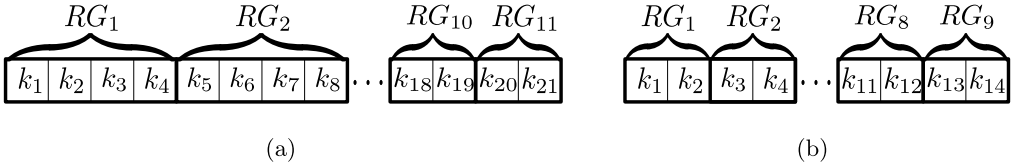

Figure 7. The structure of the chromosome holding the rate constants of (a) the weight-loop perceptron, (b) the weight-race perceptron. The rate constants (chromosome elements) are shared among reactions within the reaction group (RG) as defined in Table 3.

Table 5. The bounds that restrict the value ranges of rate constants during mutation, and the generation of initial population in GA. They are specified for three reaction types: mass-action, catalysis, and inhibition.

Type	Reaction	Rate	Rate constant bounds
Mass-action	$S \to P$	$k[S]$	$k \in [0, 0.5]$
Catalysis	$E + S \rightleftharpoons ES \to E + P$	$\frac{k_{cat}[E][S]}{K_m + [S]}$	$k_{cat} \in [0, 0.5]$
	$E + S_1 + S_2 \rightleftharpoons ES \to E + P_1 + P_2$	$\frac{k_{cat}[E][S_1][S_2]}{K_m + k_1[S_1] + k_2[S_2] + [S_1][S_2]}$	$K_m \in [0, 5]$
			$k_1, k_2 \in [0, 1]$
Inhibition	$I + S \to I + P$	$\frac{k[S]}{1 + K_i[I]}$	$K_i \in [0, 40]$

setting and constants of the GA are: population size $M = 100$, elite size $E = 20$, crossover probability $p_c = 0.9$, per-element mutation probability $p_m = 0.2$, uniform mutation strength $s_m = 0.5$, and generation limit $G = 40$.

The GA reaches solutions with fitness above 0.9 within a couple of generations, and then it continues at a slower pace toward the maximum fitness of 1, which is reached around generation 20 for both perceptrons (Figure 8). Since the learning action series are random and the simulation cost of a more precise fitness evaluation of each chromosome is too high, small fluctuations of the fitness occur even after the maximum has been reached. This is, however, not critical, since the best chromosomes found by the GA (Table 6) can learn all linearly separable logic functions with a 100% correct rate, as demonstrated in Section 7.1.

Overall, the fitness landscape of the rate constants for both perceptrons has the shape of a high plateau; hence, finding acceptable rate constants is not difficult. This demonstrates that our structural design of the chemical perceptrons in terms of species and reactions already provides correct behavior, and the perceptrons do not need to rely on the specific rate constants. In fact, we show in Section 7.2 that both models are extremely robust to the perturbation of rate constants.

We want to stress that in this article we consider the GA as just a tool, rather than an objective of our research. Since our GA setting produced satisfactory results, we have not explored optimization of the evolutionary process any further.

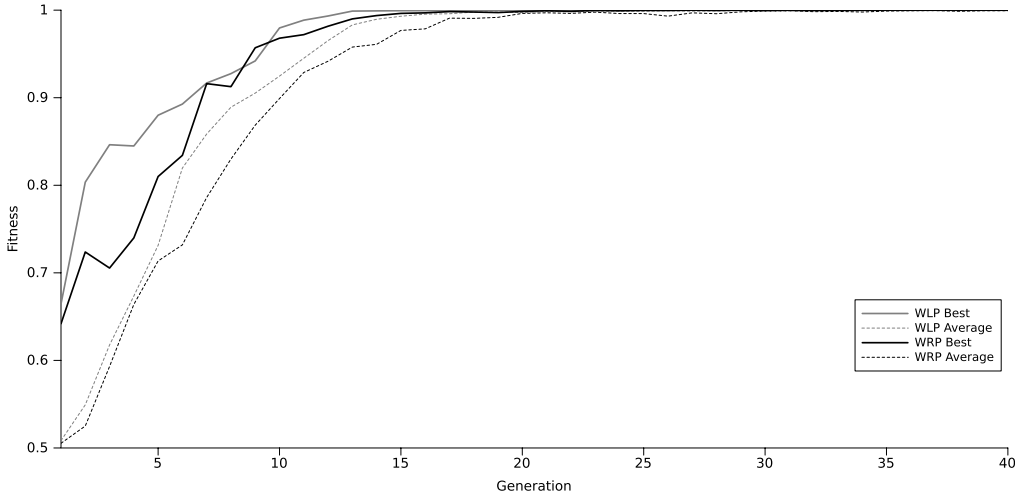

Figure 8. The best and average fitness of chromosomes representing rate constants during a single evolutionary run for: (WLP) the weight-loop perceptron, (WRP) the weight-race perceptron.

Table 6. The rate constants of the best chromosome found by the GA, rounded to four decimal places.

Weight-loop perceptron			Weight-race perceptron		
Reaction group	Rate constant	Value	Reaction group	Rate constant	Value
1	k_1	0.0838	1	k_1	0.0972
	k_2	3.7116		k_2	4.7912
	k_3	0.2686	2	k_3	0.0019
	k_4	0.4393		k_4	5.0000
2	k_5	0.1630	3	k_5	0.0081
	k_6	0.4358		k_6	3.0102
	k_7	0.5058	4	k_7	0.5000
	k_8	0.7404	5	k_8	0.5000
3	k_9	0.0974	6	k_9	0.0011
	k_{10}	4.5073	7	k_{10}	0.0132
4	k_{11}	0.0093	8	k_{11}	0.0265
	k_{12}	8.3625		k_{12}	1.8421
5	k_{13}	0.2448	9	k_{13}	0.3786
6	k_{14}	0.4249		k_{14}	0.0477
7	k_{15}	0.0115			
8	k_{16}	0.0009			
9	k_{17}	0.0018			
10	k_{18}	0.0710			
	k_{19}	0.3033			
11	k_{20}	0.5000			
	k_{21}	0.1955			

In Section 4.4 we demonstrated that the weight-race perceptron must follow the two-to-one rate constant ratio of $X \rightarrow Y$ reactions for the weights W_1 and W_2 versus the weight W_0, if the goal is to model the formal perceptron, where the weights contribute equally to the weight sum. Nonetheless, rate constants that do not obey this ratio can still perform well. Indeed, the best rate constants (chromosome) obtained by the GA (Table 6, right) makes the weight W_0 catalyze input-to-output reactions more strongly than the rest of the weights. Note that all reactions for weight species W_1 and W_2 are symmetric. The weight-race perceptron balances this preference by a lower consumption of desired output

molecules D for an adaptation of weight W_0. Still, because of the linearity and monotonicity of adaptation, the feedback loop adjusts the weights correctly regardless of the weight preference.

7 Results

7.1 Learning Performance

We evaluated the performance of both the weight-loop perceptron and the weight-race perceptron with the best rate constants found by the GA. Similarly to the fitness evaluation, the perceptrons were run against 14 action series; however, in order to achieve higher precision, this process was repeated 10^4 times as opposed to 30 times for the fitness evaluation, and each action series consisted of $L_p = 200$ as opposed to $L_f = 120$ actions (training iterations).

Figure 9 shows the weight-loop and the weight-race perceptron learning the NAND function. The training starts by the setting of weights as the constant 0 function. After 16 training iterations (8×10^4 time steps), both perceptrons successfully adapt to the NAND's input-output characteristics. Furthermore, they smoothly handle the transition from $W_0^{\ominus}$ to $W_0^{\oplus}$ species and low concentration of the output species. An interesting feature is that a chemical perceptron continues to improve its weights even after the qualitative 1, 1, 1, 0 solution is found, trying to further strengthen the output signal.

Figure 10 presents the performance for all linearly separable binary functions averaged over 14×10^4 runs. The results show that both chemical perceptrons can successfully learn 14 logic functions and reach the perfect score of 100% in all cases. The average error of each perceptron is around 6×10^{-3}% after 120 training iterations. The error occurs when the concentrations of weights with opposite signs, such as $W_0^{\oplus}$ and $W_1^{\ominus}$, are (almost) identical. Then the perceptron does not produce enough output to catalyze weight adaptations and learning slows down. However, this

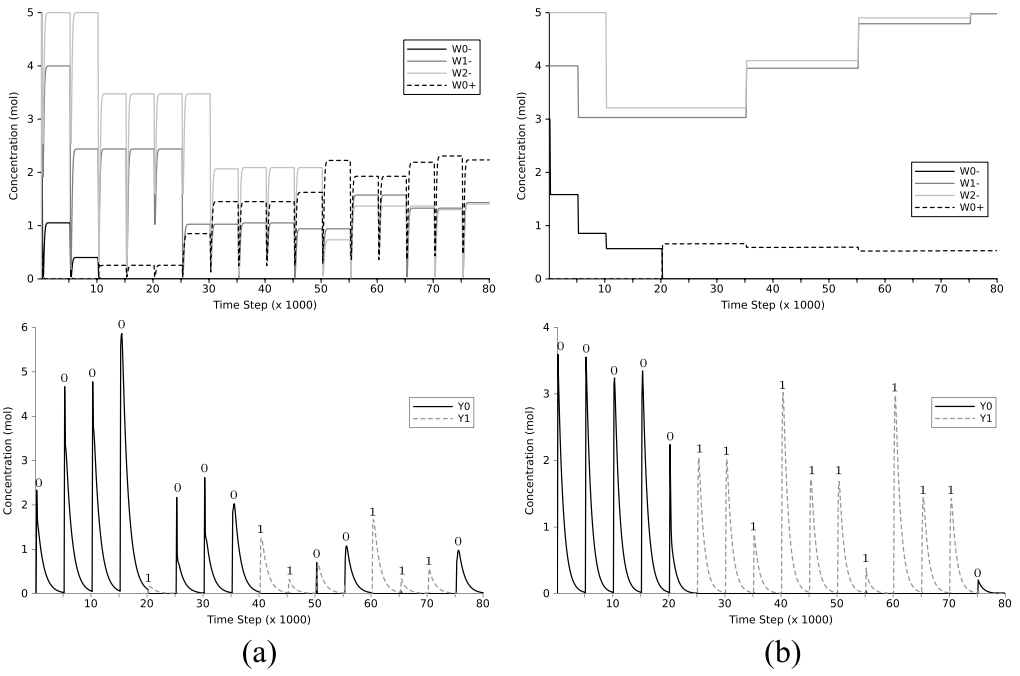

Figure 9. Training of the weight-loop perceptron (a) and the weight-race perceptron (b) to perform the NAND function starting from the CONST0 setting: top, adaptation of weights with the initial concentrations $[W_0^{\ominus}] = 3$, $[W_1^{\ominus}] = 4$, $[W_2^{\ominus}] = 5$; bottom, output of the perceptron during a training. Constant 0s gradually change to the NAND function outputs 1, 1, 1, 0. Note that we have replaced a random order of training samples by a fixed order and reduced the number of training iterations (actions) from $L_p = 200$ to 16, solely for demonstration purposes.

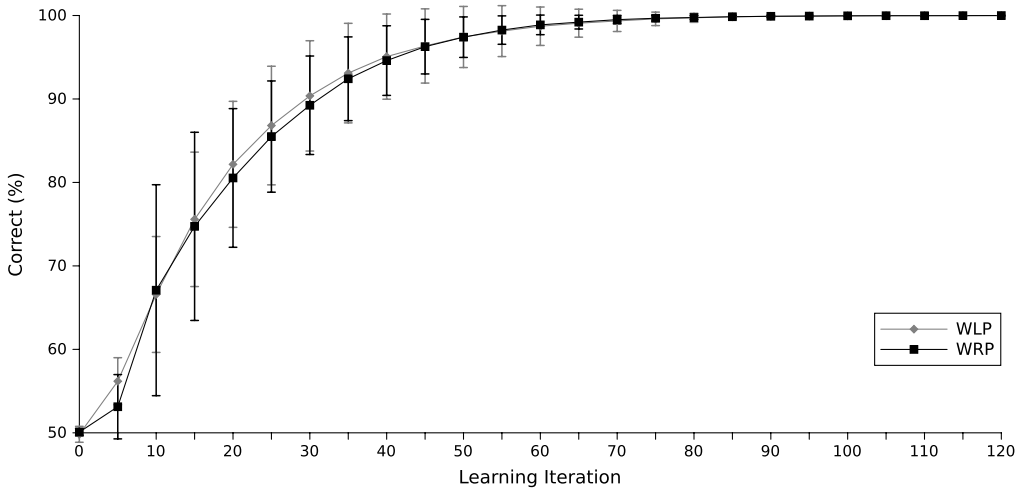

Figure 10. Mean and standard deviation of the 14 correct learning rate averages: WLP, weight-loop perceptron; WRP, weight-race perceptron. Each average corresponds to one linearly separable binary function, for which 10^4 runs were performed.

situation is rather sporadic, and after running the perceptron for 80 more learning iterations, the error disappeared.

7.2 Robustness Analysis

The chemical perceptrons perform almost identically. Moreover, their behaviors are also very similar under perturbation of the rate constants. The perturbation strength p defines how much the constants can change. Given p, each rate constant is perturbed uniformly over the interval $(0, p)$. Thus, for a perturbation $q \in (0, p)$ a rate constant γ changes to $(1 \pm q)\gamma$, where increment and decrement have equal probability. The perturbation is not limited by the bounds used for the GA (Table 5); nonetheless, when $p > 1$, a perturbation to a negative value is replaced by zero, since the rate constants cannot be negative.

We analyzed the robustness of perceptrons with the best rate constants only. As presented in Figure 11, the chemical perceptrons maintain high robustness; even for $p = 0.5$, the mean correct rate after 200 learning iterations is 98.98% for the weight-loop perceptron and 99.34% for the weight-race perceptron. The main difference between the perceptrons in robustness occurs with high perturbation strengths, at which the weight-loop perceptron becomes slightly more vulnerable. For $p = 2.0$ the performance of the weight-loop perceptron even drops below 50%, which is a sign that the concentration of output species rises beyond the maximal limit in some simulations. Recall that for the weight-loop perceptron this situation might happen due to an open fuel influx.

7.3 Model Comparison

Table 7 summarizes the attributes of our perceptron models. They perform almost identically and have very similar robustness characteristics; nonetheless, the weight-race perceptron is simpler, because it consists of fewer species and reactions and does not require any inhibition. On the other hand, the weight-loop perceptron is a closer match to the formal perceptron.

8 A Biochemical Implementation

One potential framework for a biochemical implementation of our chemical models, especially the weight-race perceptron, is DNA strand displacement [53]. The use of DNA allows one to pick arbitrary sequences to stand for an arbitrary species from an artificial chemistry. In strand displacement

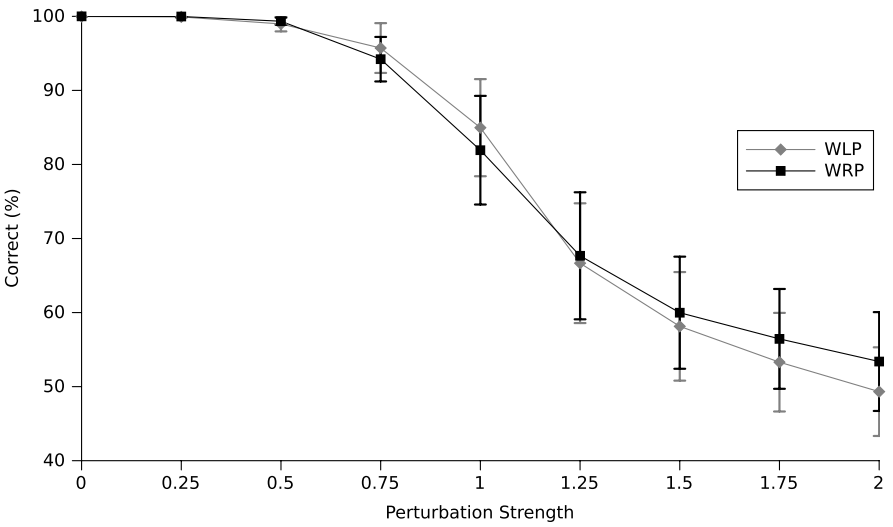

Figure 11. Mean and standard deviation of the 14 final correct rate averages under the perturbation of rate constants after 200 learning iterations: WLP, weight-loop perceptron; WRP, weight-race perceptron. Each average corresponds to one linearly separable binary function, for which 10^4 runs were performed. Given perturbation strength p, each rate constant is perturbed uniformly over the interval $(0, p)$.

systems, populations of these species are typically represented by the populations of single-stranded DNA molecules. These interact with double-stranded gate complexes, which mediate transformations between free signals. Soloveichik et al. [46] proved that a strand displacement circuit can approximate, with arbitrarily small error, any artificial chemistry based solely on mass-action kinetics. This offers a way to derive a DNA implementation of our perceptron models. First, we would need to translate the whole model into mass-action kinetics. The catalytic reactions that we previously modeled using Michaelis-Menten kinetics could be handled by expanding out the full set of enzymatic reactions. On the other side, the uncompetitive inhibitory reactions cannot be directly represented by mass action and therefore must first be adapted to the competitive version. Because of this complication, the weight-race perceptron, which does not use an inhibition, is more suitable for mass-action transformation and thus also for DNA strand displacement implementation.

An expanded mass-action version of the weight-race perceptron would undoubtedly be larger than the model presented in this article. We estimate it would consist of 40–50 species and 50–60 reactions,

Table 7. The comparison of the weight-loop perceptron and the weight-race perceptron.

Attribute	WLP	WRP
Number of species	21	**14**
Number of reactions	34	**30**
Catalysis needed	Yes	Yes
Inhibition needed	Yes	**No**
Average learning performance	**100%**	**100%**
Rate robustness (50% perturbation)	**98.98%**	**99.34%**
Follows the formal perceptron	**Yes**	No

which is high but still manageable. Moreover, we might be able to simplify the encoding, for example by modeling the ⊕ and ⊖ and the 0 and 1 variants of species as two complementary strands, where annihilation is actually the production of an inert double-stranded complex. The state of the art in strand displacement circuits is a four-bit square root circuit [52] including 130 strands (74 non-input species). The number of species is higher than we need for our perceptron, but the square root circuit was built with seesaw gates, which are much simpler than the gates in Soloveichik's encoding. While the expanded model of a chemical perceptron would still be impractical for implementation in the laboratory in the near future, it would at least demonstrate that our model could in principle be implemented using real DNA strands. Moreover, the high robustness of the perceptron models to rate constant perturbations means that precisely reproducing the optimal rate constants might not be necessary in a strand displacement implementation.

9 Discussion and Further Research

There are several potential areas for improvement. First, instead of the fast but rather coarse-grained Euler method for the numerical integration of ODEs, the chemical model simulation could incorporate the more accurate Runge-Kutta approximation [48], or possibly some stochastic method [29, 51]. Similarly, lowering the time step would deliver higher-precision results, but at larger simulation costs.

All reactions in our model are irreversible, which simplifies reasoning about their causality. However, a reversible reaction with rates based on equilibrium of the forward and the backward part is more plausible and could cover an irreversible reaction as a special case. Also, the structural aspects of molecular species as an intermediate level between artificial and real chemistry would be worth investigation.

Since the weight-race perceptron excludes inhibition and in it all catalytic reactions with the exception of weight adaptation require only one catalyst, it corresponds more closely to real chemistry. We argue that after small adjustments, the reactions with two dependent catalysts, in which an input species and an output species together catalyze the weight adaptation, can broaden to the standard one-catalyst Michaelis-Menten kinetics. Then, the model can be further adapted, so it follows pure mass-action kinetics, where a catalysis is expanded to partial association and disassociation reactions.

Our AC model employs three concentration thresholds. The lower threshold avoids numerical instability by cutting the concentration to zero if it goes below 10^{-3} M. The upper threshold of 50 M detects whether the concentration of a species diverges beyond bounds (explodes). The difference threshold, 10^{-5} M, defines the minimal measurable change in the concentration of a species. If the concentration of a species increases or decreases very slowly, it is considered as a constant. Then if the concentration of every species is constant, the system is at the fixed point, and hence the AC run can proceed directly to the next input (action), and the AC simulation cost can be reduced.

To make the system more biologically realistic, we can wrap a perceptron in a permeable compartment, which provides integrity, controls input-output streams, and avoids duplication of species in case the chemical system consists of multiple perceptrons. Then an efflux of obsolete, noninteracting product might serve as a decay reaction.

The chemical perceptrons do not age; hence, the learning rate (or the weight-adaptation strength) remains constant through the whole learning process. A decrease of the learning rate over time usually improves convergence. In a chemical system we cannot assume that the perceptron knows when training starts and ends; therefore, if we assumed that the perceptron ages, this process would need an inner quality of the perceptron that degrades (decays) over time.

Our perceptron setup might present a modularity problem in the case of connecting multiple perceptrons together—the output from a perceptron in the first layer would not necessarily be at an appropriate level to feed into a perceptron in the second layer, since the input concentrations are fixed but the output concentrations are not. As a consequence, the execution setup with fixed interval length between actions and with constant concentration of input and desired-output species could have variations in order to obtain perceptrons with more generality, or a smaller bias to the specific concentration range. Also, it would be interesting to address the learning capabilities of the

system in the face of noise in the training instances provided. We could introduce noise into reaction rates similarly to that in our robustness analysis, but at each step during the simulation rather than just initially. Finally, it would be worthwhile to explore repair capabilities of a chemical perceptron from the perspective of systems biology or relational biology. Decay of currently stable weight species would compel a chemical perceptron to repair its inner state perpetually by consuming an external resource.

10 Conclusion

We have demonstrated a proof of concept of learning and adaptation in an artificial chemical system. Artificial chemistry provided a conceptual framework, using sets of species, reactions, and reaction rates for modeling the functional characteristics of a two-input binary perceptron. After each processing, the perceptron recovers its default ready state, so it is reusable without outside intervention. We have introduced two models—the weight-loop perceptron and the weight-race perceptron. The weight-loop perceptron calculates the weight sum directly by transforming the weights to output and then recovering them to the original concentration. In the weight-race perceptron, the weights catalyze the input-to-output reactions; hence, the weight sum (the output) is calculated indirectly by weight competition (rate comparison).

Both chemical perceptrons can be trained perfectly to a desired logic function by providing sufficient input-output training pairs. Furthermore, they maintain high robustness to the perturbation of rate constants and achieve a 99% success rate for a 50% perturbation. Therefore, an implementation of our perceptrons in real chemistry would not have to use our specific rate constants in order to perform well. Since the weight-race perceptron's reactions are simpler and directly transformable to pure mass-action model, it is more practical and suitable for real chemical experimentation.

By implementing the mass-action model as a DNA-strand-displacement circuit, the chemical perceptron can serve as the basis of a new abstract layer for biochemical computing. Such a programming interface would hide inner chemical "hardware," so we could program it to the desired function without knowing anything about chemistry.

Acknowledgments

This research was funded by NSF grant 1028120.

References

1. Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science*, *266*(5187), 1021–1024.
2. Amos, M. (2003). *Theoretical and experimental DNA computation*. Berlin: Springer-Verlag.
3. Arnaut, L. G., Formosinho, S. a. J., & Burrows, H. (2007). *Chemical kinetics: From molecular structure to chemical reactivity*. Amsterdam: Elsevier.
4. Banzhaf, W. (1990). The molecular traveling salesman. *Biological Cybernetics*, *64*(1), 7–14.
5. Bitbol, M., & Luisi, L. P. (2004). Autopoiesis with or without cognition: Defining life at its edge. *Journal of The Royal Society Interface*, *1*(1), 99–107.
6. Braich, R. S., Chelyapov, N., Johnson, C., Rothemund, P. W. K., & Adleman, L. (2002). Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, *296*, 499–502.
7. Bray, D. (1995). Protein molecules as computational elements in living cells. *Nature*, *376*(6538), 307–312.
8. Brooks, R. (1989). A robot that walks; Emergent behaviors from a carefully evolved network. *Neural Computation*, *1*(2), 253–262.
9. Buchler, N. E., Gerland, U., & Hwa, T. (2003). On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences of the United States of America*, *100*(9), 5136–5141.
10. Copeland, R. A. (2002). *Enzymes: A practical introduction to structure, mechanism, and data analysis* (2nd ed.). New York: Wiley.

11. de Silva, A. P., Dobbin, C. M., Vance, T. P., & Wannalerse, B. (2009). Multiply reconfigurable "plug and play" molecular logic via self-assembly. *Chemical Communications*, No. 11, pp. 1386–1388.

12. Dittrich, P. (2005). Chemical computing. In J.-P. Banâtre, P. Fradet, J.-L. Giavitto, & O. Michel (Eds.), *Unconventional programming paradigms* (pp. 21–32). Berlin: Springer-Verlag.

13. Dittrich, P., Ziegler, J., & Banzhaf, W. (2001). Artificial chemistries—A review. *Artificial Life*, *7*(3), 225–275.

14. Elliott, L., Ingham, D., Kyne, A., Mera, N., Pourkashanian, M., & Wilson, C. (2004). Genetic algorithms for optimisation of chemical kinetics reaction mechanisms. *Progress in Energy and Combustion Science*, *30*(3), 297–328.

15. Epstein, I. R., & Pojman, J. A. (1998). *An introduction to nonlinear chemical dynamics: Oscillations, waves, patterns, and chaos*. Oxford, UK: Oxford University Press.

16. Espenson, J. (1995). *Chemical kinetics and reaction mechanisms*. New York: McGraw-Hill.

17. Faulhammer, D. (2000). Molecular computation: RNA solutions to chess problems. *Proceedings of the National Academy of Sciences*, *97*(4), 1385–1389.

18. Gillespie, D. T. (1976). A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, *22*(4), 403–434.

19. Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, *81*(25), 2340–2361.

20. Gutiérrez-Naranjo, M. A., & Pérez-Jiménez, M. J. (2009). In *Membrane computing* (pp. 217–230). Berlin: Springer-Verlag.

21. Haykin, S. (2009). *Neural networks and learning machines* (3rd ed.). Upper Saddle River, NJ: Prentice Hall.

22. Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.

23. Hinze, T., Fassler, R., Lenser, T., Matsumaru, N., & Dittrich, P. (2009). *Membrane computing* (pp. 231–245). Berlin: Springer-Verlag.

24. Hjelmfelt, A., & Ross, J. (1992). Chemical implementation and thermodynamics of collective neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, *89*(1), 388–391.

25. Hjelmfelt, A., Weinberger, E. D., & Ross, J. (1991). Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences of the United States of America*, *88*(24), 10983–10987.

26. Holmes, M. H. (2007). *Introduction to numerical methods in differential equations*. Berlin: Springer-Verlag.

27. Ionescu, M., Paun, G., & Yokomori, T. (2006). Spiking neural P systems. *Fundamenta Informaticae*, *71*(2), 1–28.

28. Ionescu, M., & Sburlan, D. (2008). Some applications of spiking neural P systems. *Computing and Informatics*, *27*(3), 515–528.

29. Jahnke, T., & Altntan, D. (2010). Efficient simulation of discrete stochastic reaction systems with a splitting method. *BIT Numerical Mathematics*, *50*(4), 797–822.

30. Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., & Wanq, A. (1990). *Evolution as a theme in artificial life: The Genesys/Tracker system* (Technical report). Los Angeles: Computer Science Department, University of California.

31. Jonoska, N. (2008). Biomolecular automata. In O. Shoseyov & I. Levy (Eds.), *NanoBioTechnology* (pp. 267–299). New York: Humana Press.

32. Kim, J., Hopfield, J. J., & Winfree, E. (2004). Neural network computation by in vitro transcriptional circuits. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in Neural Information Processing Systems*, Vol. 17 (pp. 681–688). Cambridge, MA: MIT Press.

33. Libecq, C. (Ed.). (1992). *Biochemical nomenclature and related documents: A compendium* (2nd ed.). London: Portland Press.

34. Maes, P. (1994). Modeling adaptive autonomous agents. *Artificial Life*, *1*, 135–162.

35. Michaelis, L., & Menten, M. L. (1913). Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, *49*, 333–369.

36. Mills, A. P., Yurke, B., & Platzman, P. M. (1999). Article for analog vector algebra computation. *Biosystems*, *52*(1–3), 175–180.

37. Minsky, M., & Seymour, P. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

38. Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

39. Mitchell, M., & Forrest, S. (1994). Genetic algorithms and artificial life. *Artificial Life*, *1*(3), 267–289.

40. Nahler, G. (2009). Michaelis-Menten kinetics. In *Dictionary of pharmaceutical medicine* (pp. 113–113). Berlin: Springer-Verlag.

41. Ouyang, Q. (1997). DNA solution of the maximal clique problem. *Science*, *278*(5337), 446–449.

42. Pei, R., Matamoros, E., Liu, M., Stefanovic, D., & Stojanovic, M. N. (2010). Training a molecular automaton to play a game. *Nature Nanotechnology*, *5*(11), 773–777.

43. Polifke, W., Geng, W., & Döbbeling, K. (1998). Optimization of rate coefficients for simplified reaction mechanisms with genetic algorithms. *Combustion and Flame*, *113*(1–2), 119–134.

44. Rojas, R. (1996). *Neural networks: A systematic introduction*. Berlin: Springer-Verlag.

45. Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review*, *65*, 368–408.

46. Schnell, S., & Mendoza, C. (1997). Closed form solution for time-dependent enzyme kinetics. *Journal of Theoretical Biology*, *187*, 207–212.

47. Soloveichik, D., Seelig, G., & Winfree, E. (2010). DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences of the United States of America*, *107*(12), 5393–5398.

48. Stoer, J., & Bulirsch, R. (2002). *Introduction to numerical analysis*. Berlin: Springer-Verlag.

49. Stojanovic, M. N., & Stefanovic, D. (2003). A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, *21*(9), 1069–1074.

50. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

51. Turner, T. E., Schnell, S., & Burrage, K. (2004). Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, *28*(3), 165–178.

52. Qian, L., & Winfree, E. (2011). Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, *332*, 1196–1201.

53. Zhang, D. Y., & Seelig, G. (2011). Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, *3*(2), 103–113.