Supervised Learning in a Multilayer, Nonlinear Chemical Neural Network

David Arredondo and Matthew R. Lakin, *Member, IEEE*

***Abstract*— The development of programmable or trainable molecular circuits is an important goal in the field of molecular programming. Multilayer, nonlinear, artificial neural networks are a powerful framework for implementing such functionality in a molecular system, as they are provably universal function approximators. Here, we present a design for multilayer chemical neural networks with a nonlinear hyperbolic tangent transfer function. We use a weight perturbation algorithm to train the neural network which uses a simple construction to directly approximate the loss derivatives required for training. We demonstrate the training of this system to learn all 16 two-input binary functions from a common starting point. This work thus introduces new capabilities in the field of adaptive and trainable chemical reaction network (CRN) design. It also opens the door to potential future experimental implementations, including DNA strand displacement reactions.**

***Index Terms*— Chemical reaction networks (CRNs), hyperbolic tangent, neural networks, nonlinearity.**

I. Introduction

THE field of molecular programming lives at the intersection of computer science and wet chemistry. A key goal in this field is to interface rationally designed information processing systems with biology. Engineered nucleic acids can implement computational systems capable of reproducing the behavior of conventional algorithms, using biomolecules as inputs, outputs, and computational components. Such systems have been designed to implement a range of computational frameworks, including digital logic circuits [1], [2], signal amplifiers [3], distributed algorithms [4], localized molecular computations [5], molecular oscillators [6], and intracellular computations [7].

Here, we focus on the implementation of algorithms via reactions between abstract chemical species, which could, in principle [8], be implemented using toehold-mediated DNA strand displacement [9]. In molecular programming, the requirement to redesign a new molecular circuit for each different computation hinders the pace of development. Therefore, the ability to implement programmable or "trainable" molecular circuits with adaptive behavior would significantly enhance the state of the art. The contribution of this article is a design for such a system: a chemical reaction network (CRN) that implements a trainable autonomous artificial neural network with a mathematically well-defined learning algorithm.

Molecular computing systems that react in real time to changing biomolecular inputs, such as blood glucose levels [10] or mutating viral genomes in biomedical diagnostics [11], would be a valuable tool for the biomedical sciences. Previous molecular implementations of neural networks [12], [13] can be configured to yield a desired output for a given input; however, they must be trained *in silico* because they lack the feedback mechanisms necessary for autonomous learning within the chemical system itself. To date, proposals for autonomous learning systems fall short in that they only qualitatively implement the behavior of a neural network [14]. Other proposed systems cannot learn nonlinear functions such as XOR [15]–[17].

We present a CRN that implements the stochastic gradient descent algorithm to train a multilayer, nonlinear artificial neural network. Our network architecture is multilayer because it has two layers: a hidden layer (containing two neurons) and an output layer (containing one neuron). The stochastic gradient descent algorithm is used to train the neural network to reproduce the behavior of a particular function. Using computational simulations of a deterministic kinetic model derived from the CRN, we show that the network can be trained to implement any of the 16 two-input Boolean logic functions via supervised learning [14], [17]. Training examples consist of two binary input signals and the respective binary output. In each training round, the desired output is compared to the output of the network given its current weights. The partial derivative of the loss (output error) with respect to a particular weight is used to update that weight value in the network so as to reduce the loss in subsequent rounds.

Our network is nonlinear because each artificial neuron incorporates a nonlinear transfer function. Specifically, we use the hyperbolic tangent, a continuous sigmoid that squashes its input into the interval $(-1, 1)$ and that is of practical interest in mainstream machine learning research. We report a novel construction that allows the hyperbolic tangent of an input signal to be computed exactly using a small set of abstract chemical reactions, thereby enabling a mathematically precise definition of the behavior of our CRN.

Manuscript received 22 June 2021; revised 15 September 2021 and 26 October 2021; accepted 19 January 2022. Date of publication 8 February 2022; date of current version 6 October 2023. This work was supported by the National Science Foundation under Grant 1935087, Grant 1814906, Grant 1525553, and Grant 1518861. *(Corresponding author: Matthew R. Lakin.)*

David Arredondo is with the Center for Biomedical Engineering, The University of New Mexico, Albuquerque, NM 87131 USA.

Matthew R. Lakin is with the Department of Computer Science, Department of Chemical & Biological Engineering, and the Center for Biomedical Engineering, The University of New Mexico, Albuquerque, NM 87131 USA (e-mail: mlakin@cs.unm.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3146057.

Digital Object Identifier 10.1109/TNNLS.2022.3146057

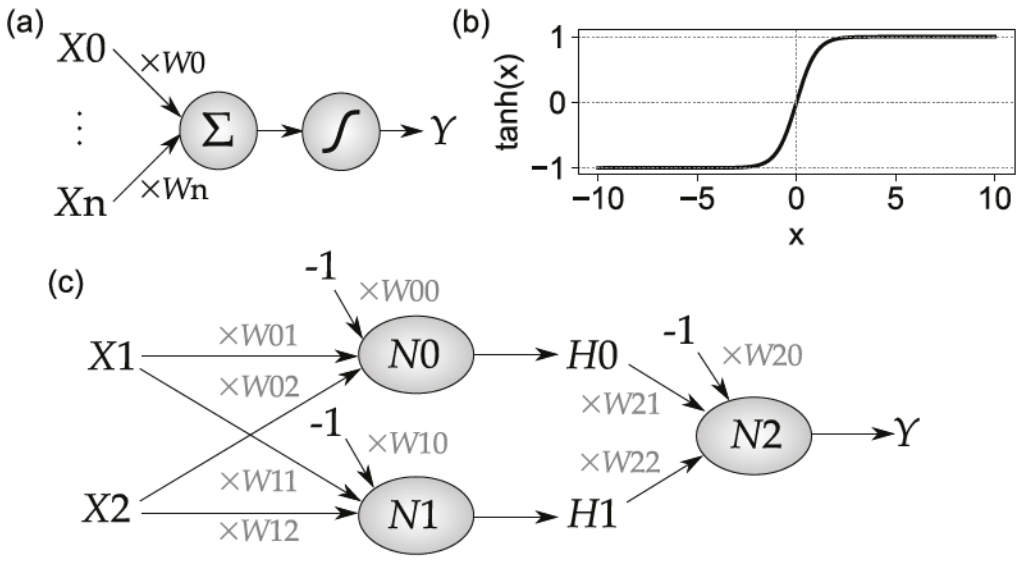

Fig. 1. Architecture of our artificial neural network system. (a) A single artificial neuron computes a weighted linear sum of its inputs and then passes the result of that sum through a nonlinear transfer function to produce its output value. (b) Graph of the tanh transfer function, which we use in this work, over the interval $(-10, 10)$. The tanh function is a sigmoid that squashes its input into the interval $(-1, 1)$. (c) The particular artificial neural network instance under study here. There are two input signals, $X1$ and $X2$. These are fed into the two "hidden layer" neurons, $N0$ and $N1$. The outputs of the two hidden layer neurons, $H0$ and $H1$, feed into a single "output layer" neuron, $N2$, which produces the final output signal Y. Each neuron has a total of three inputs, the third being a "bias" signal which is always set to -1. Each input signal to each artificial neuron has its own weight, thus there are nine weights in total. We write Wij for the weight associated with the jth input to neuron Ni.

A key aspect of the gradient descent algorithm is computing the derivative of the loss function. We address this issue by directly approximating the partial derivative of the loss function by finite difference approximation using weight perturbation [18]. For each training example, another output is computed in parallel by a copy of the network with a small change in one of its weights, and the difference between these outputs is proportional to the partial derivative of the loss function with respect to the given weight. We show that from the same set of initial weight values, all 16 binary operators are learned by the CRN in ten rounds of training and also generalize to a highly parallel construction in which all of the weights are perturbed in parallel. Our work thus provides a novel design for a learning CRN that could find application in adaptive and trainable molecular computing systems.

II. Results

The structure of our CRN is shown in Fig. 1, which shows the basic architecture of an artificial neuron [see Fig. 1(a)], a plot of the hyperbolic tangent (tanh) function that we will use as the nonlinear transfer function in our artificial neurons [see Fig. 1(b)], and the architecture of the neural network that we will implement as a CRN and train [see Fig. 1(c)]. This architecture takes two inputs ($X1$ and $X2$) that are both passed to two hidden-layer neurons ($N0$ and $N1$). The outputs ($H0$ and $H1$) of the hidden layer neurons feed into a single output layer neuron ($N2$), whose output (Y) is the overall network output. Each neuron also has a third "bias" input whose value is always -1. This neural network architecture is simple and yet powerful enough to learn all of the two-input binary functions, including those that are not linearly separable, such as XOR [14].

To simplify the presentation, we break down the network into logical constituent parts, which we describe separately. We then present simulation results that demonstrate mathematically faithful implementation of our weight perturbation training algorithm for artificial CRN neurons using the practically relevant tanh function as their nonlinear transfer function. We show this both for the simple case where a single weight is perturbed in each training round and for the generalization that can update all of the weights in parallel.

A. CRN Model and Simulation Process

We use a deterministic ordinary differential equation (ODE) model of chemical reaction kinetics. We assume that reactants and products of individual reactions may be arbitrary multisets of chemical species, that is, reactions with arbitrary numbers of reactants and products are permitted. We use standard mass action kinetics and our simulation approach is to use the mass action rate law to convert the abstract CRN into a collection of ODEs. We then carry out deterministic simulations of network dynamics using a simulator written in Python that we call ProBioSim, which constructs the ODEs from the CRN notation and integrates them. Importantly, the ProBioSim simulator provides the ability to perturb the system after time $t = 0$, which corresponds to the subsequent addition of species representing the training instances. This simulator is available online from: https://github.com/matthewlakin/ProBioSim/.

B. Learning CRN Architecture

We begin by discussing the version of the system in which a single weight is updated during each training round. The high-level architecture of this learning CRN is presented in

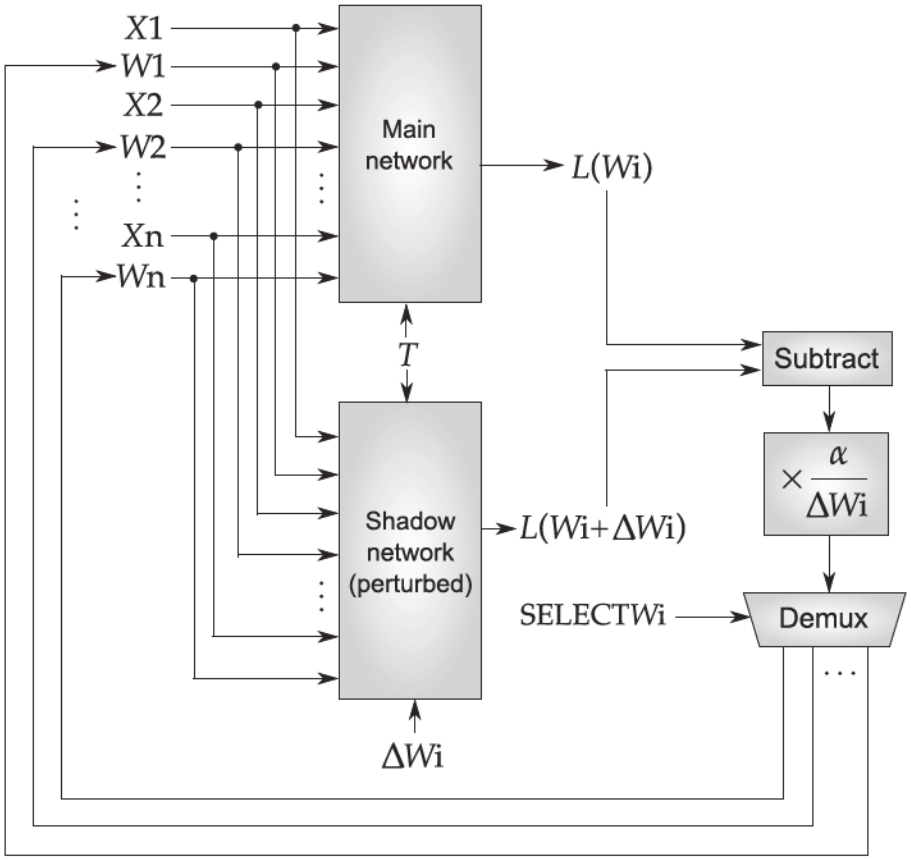

Fig. 2. Overview of chemical reaction network architecture for training an artificial neural network via weight perturbation. The input signals Xi and associated weights Wi, are inputs to the system. For supervised learning, the target output value T must also be supplied. These inputs are duplicated into two copies of the network, the main network and the shadow network, where the shadow network has one weight Wi perturbed by an amount ΔWi. Both networks use the input values, weight values, and target values to compute the loss incurred by the network relative to that target value. All of these signals are represented as concentrations of species in an abstract CRN. For the main network, the value of the loss species represents when weight Wi is unperturbed, that is, $L(Wi)$. For the perturbed network, the value of its loss species represents the loss when Wi is perturbed by an amount ΔWi, that is, $L(Wi + \Delta Wi)$, that is, the loss under the perturbed version of the weights. These loss values are subtracted and then multiplied by $(\alpha/\Delta Wi)$ to produce the corresponding weight update amount for the perturbed weight Wi. This is then fed through a demultiplexer, which uses a "selector" species SELECTWi to determine which of the weights that value will be fed back into. This feedback process modifies the concentrations of the weight species so that the next training round will take place starting from those modified weights and can then modify them further.

Fig. 2. The network is actually implemented twice, by similar CRNs containing distinct abstract chemical species. We refer to these as the "main" and "shadow" networks. We append the prime symbol (or Prime) to the shadow network species names to distinguish them from the corresponding species in the main network. For example, the shadow network counterpart of a species X would be X'. The weights are stored as the concentrations of certain species in the main and shadow networks, which are kept in sync by the design of the circuit. The inputs at each supervised training round are the input values and the expected ("target") output value. These first undergo a fan-out process that duplicates them into both the main and shadow networks with equal values in each. The main network computes the output of the network, and the associated loss L, according to the current stored weight values and these presented inputs, according to $L = (Y - T)^2$, where Y is the output from the network and T is the target (expected) output. The shadow network, on the other hand, has one of its weights (Wi say) perturbed by a small amount ΔWi, so its computed loss value will be slightly different. This perturbation is achieved in the CRN by adding the corresponding additional amount of the Wi' that represents that weight in the shadow network. By comparing the different loss values computed by the main and shadow networks, the first derivative of the loss with respect to the perturbed weight can then be estimated, as outlined below. This value is then fed back to update the selected weight, which completes the training round. We now present the detail of how the different parts of the circuit are implemented as an abstract CRN.

C. Timing Reactions Using a Molecular Clock Signal

A key aspect of implementing a neural network CRN is timing chemical reactions to occur in the correct order. We follow the approach of Vasic *et al.* [19], using a molecular oscillator to produce a "clock signal" that controls the availability of catalyst species Ci that catalyze the other reactions, so they can only occur with a non-negligible rate during the clock phase where that clock signal is high. This produces a system that runs as autonomously as possible, relying only on the inputs provided at the start of each training round to drive the computation. Our clock reactions are as follows:

TABLE I
DIVISION OF LEARNING CRN OPERATIONS INTO CLOCK PHASES

Clock phase	Reactions
$C1$	Fan-out inputs to main and shadow network(s)
$C3$	Compute weighted sums for hidden layer neurons ($N0$, $N1$)
$C5$	Compute tanh transfer function for hidden layer neurons ($N0$, $N1$)
$C7$	Compute weighted sum for output layer neuron ($N2$)
$C9$	Compute tanh transfer function for output layer neuron ($N2$)
$C11$	Compute loss in main and shadow network(s)
$C13$	Compute estimate of loss derivative(s)
$C15$	Feed back loss derivative estimate(s) to update weight(s)
$C17$	Reset for next round by degrading certain species

1) $C1 + C2 \xrightarrow{\text{kClock}} C2 + C2$
2) $C2 + C3 \xrightarrow{\text{kClock}} C3 + C3$
3) ...
4) $C20 + C1 \xrightarrow{\text{kClock}} C1 + C1$.

We use an oscillator with 20 phases to serve as our clock. We only use the odd-numbered clock signals to catalyze the other reactions in the system, so that the different phases of execution do not overlap. In our simulations, we set the kClock rate constant to 0.1. The clock signals $C19$ and $C20$ are initially set high, with concentration 1.0, and all other clock signals are initially set low, with a concentration of 10^{-6}. These settings give a clock phaselength of ~72.9 time units, which empirically is sufficient to carry out the required reactions for our design in each clock phase. This value is used in the timing of the perturbations required to carry out learning, as outlined below, since the training examples and weight perturbation species must be introduced at the start of the $C1$ clock cycle. The division of the operations of our learning CRN into different clock phases is outlined in Table I.

D. Reaction Rates

All reactions other than the clock reactions will have rate constant $k = 1.0$. The correctness of our CRN does not depend on the precise value of k. However, many of the reactions rely on competitive processes to compute the correct result, meaning that the rate constants for those reactions should be the same as those with which they compete. Since our CRN design is highly abstract, we assume arbitrary units of time and concentration when specifying initial conditions and rate constant values. As such, we do not state these units explicitly and, in an abuse of notation, we use the same rate constant symbol for reactions that have different numbers of reactants and whose rate constants would thus have different units. Here, we really just mean that the rate constants for those reactions have the same numeric value.

We note that, for the correctness of our system, all of the reactions do not necessarily need to have the same rate constant value; only in the case where particular reactions are in direct competition over reactants, such as in the catalytic multiplication motif outlined below, do those particular reactions need matching rate constants. If competitive reactions have distinct rate constants and these are known, then the discrepancy can be compensated for by adjusting the input amounts. If not, then the rate mismatch will introduce errors into the result of those reactions in proportion to the degree of mismatch.

E. Dual-Rail Species Representations and Annihilation Reactions

Most of the signals involved in our system could be either positive or negative. However, the concentrations of chemical species can only be positive. Therefore, to represent these signals, we use a dual-rail representation [20], whereby each signal (X say) is actually represented by two chemical species: Xp representing the positive component and Xm representing the negative component. Then, the value of the X signal at any point is actually given by the difference between these concentrations, $[Xp] - [Xm]$. We include an annihilation reaction between the positive and negative components of every pair of dual-rail signal species in our system so that only the positive or negative component of a given signal will be present with non-negligible concentration. For example, the annihilation reaction for the dual-rail signal X would be as follows:

1) $Xp + Xm \xrightarrow{k} .$

Henceforth, we will assume that signals are dual-rail unless stated otherwise.

F. Fan-Out of Input Signals

To ensure that the input and target output signals are represented with the same values in both the main and shadow networks, each of the signals is presented as a single species. These then undergo fan-out reactions where they are consumed and converted into two new signals corresponding to the inputs and targets for the main and shadow networks. Reactions are provided to duplicate both the positive and negative dual-rail components of these input species, and these are all driven by clock signal $C1$ as they are the first thing that must occur in each training round. For example, the two reactions that handle the fan-out of the $X1$ input signal are as follows:

1) $C1 + X1p \xrightarrow{k} C1 + X01p + X11p + X01p' + X11p'$
2) $C1 + X1m \xrightarrow{k} C1 + X01m + X11m + X01m' + X11m'$.

Note that each species is duplicated twice for both for main and shadow networks, because the $X1$ signal serves as an input to neuron $N0$ as $X01$ and to neuron $N1$ as $X11$. The reactions for $X2$ are similar, and those for *BIAS* duplicate the signal three times for each network, as this signal (always having value -1) is fed into all three neurons as the bias term. In short, execution of the fan-out reactions prepares both the main and shadow networks to carry out computations in parallel on their own copies of the input signals.

G. Computing Weighted Sums

The first step of computing the output signal for a given neuron is to compute the weighted sum of its inputs. Here, we use an approach inspired by previous work [17], [21]. The weight and input species together catalyze production of the output species, and the input degrades with the same rate constant value. Thus, if the weight species is W, the input species is X, and the output species is Y, the basic scheme is as follows:

1) $W + X \xrightarrow{k} W + X + Y$
2) $X \xrightarrow{k} .$

Following our previous work [17], solving the differential equations at steady state gives the final concentration of Y as the product of the concentrations of W and X, as required. Importantly, the input signal X is consumed in this process. We use this basic reaction motif to scale all the input signals by their corresponding weights. For example, for neuron $N0$, the input BIAS0 is scaled by weight $W00$, the input $X01$ is scaled by weight $W01$, and input $X02$ is scaled by weight $W02$. The initial weight values are set in the initial conditions of the CRN, as outlined below, and are updated in response to training data. The input signals are provided as perturbations at the start of each training round, as outlined below. We use clock species $C3$ as a catalyst for all of the input weighting reactions, for neurons $N0$ and $N1$. Similar reactions are included for the output layer neuron, $N2$, catalyzed by clock species $C7$ so they occur after the hidden layer neurons have finished computing their outputs (see Table I).

H. Computing Nonlinear Transfer Function

Having computed the net value for each neuron, we must then pass it through a nonlinear transfer function. Here, we choose the hyperbolic tangent (tanh) function. Previous experimental implementations of neural networks using DNA strand displacement [2] have used threshold-based nonlinearities, and previous proposals for learning CRNs [14] have used other nonlinearities that do not correspond to a well-understood transfer function. Furthermore, the hyperbolic tangent is relevant in mainstream machine learning research. Here, we draw inspiration from the work of Fages *et al.* [22] to compute the exact value of the hyperbolic tangent for each input net value.

In previous work, Fages *et al.* [22] observed that the derivative of $\tanh(t)$ with respect to t can be expressed in terms of $\tanh(t)$ itself, as follows:

$$\frac{d\tanh(t)}{dt} = 1 - \tanh^2(t). \tag{1}$$

They then noted that a CRN could be created whose corresponding set of ODEs gives (1), since if we consider the following abstract chemical reactions:

1) $\xrightarrow{k} T$
2) $T + T \xrightarrow{k} T$

with $k = 1.0$, then we get $(dT/dt) = 1 - T^2$. Therefore, when initialized with $[T] = 0$ at time 0, the timecourse of $[T]$ traces out the tanh function, so $[T]$ at time t is equal to $\tanh(t)$.

However, we wish to convert an input concentration signal into a transformed output concentration signal at steady state. Our contribution in this regard is to note that we can, in effect, "freeze" the above reaction at the corresponding time point by providing the input signal as the concentration of an additional species that catalyzes the two reactions shown above. Then, if that catalyst can also degrade, the initial quantity of the catalyst present determines, in effect, how far along the tanh timecourse can be reached before the catalyst runs out. In the case of the neuron $N0$, we use NET0 (the output from the weight computation step) as the catalyst and the species $H0$ to represent the output. We include the clock signal $C5$ as an additional catalyst for all of these reactions to ensure that these reactions execute after the weighting reactions (in clock phase 3) have finished, giving the following reactions for the positive arm of the tanh transfer function:

1) $C5 + \text{NET0}p \xrightarrow{k} C5 + H0p + \text{NET0}p$
2) $C5 + \text{NET0}p + H0p + H0p \xrightarrow{k} C5 + H0p + \text{NET0}p$
3) $C5 + \text{NET0}p \xrightarrow{k} C5.$

This works because the value of the $H0$ signal is zero initially. Furthermore, this output signal will be consumed by the downstream reactions, thereby resetting it to zero for the next training round.

Since tanh is an odd function, that is, $\tanh(-x) = -\tanh(x)$, similar reactions involving NET0m and $H0m$ implement the negative arm of the transfer function. The $H0$ output signal then serves as one of the inputs to the output neuron, $N2$. We replicate these reactions to compute the nonlinear transfer functions for the other two neurons, $N1$ and $N2$, producing output signals $H1$ and Y, respectively. Y is the overall output signal from the neural network and the input to the loss computing module, outlined below.

I. Computing Loss Function

The goal of our learning circuit is to minimize the loss (error) between the output signal Y from the neural network and a "target" output value T, which is provided as a dual-rail signal at the start of each training round. The module of the CRN controlled by clock signal $C11$ computes this loss, which we compute as $L = Z^2$, where $Z = Y - T$. The subtraction operation $(Z = Y - T)$ is carried out by transferring the dual-rail species corresponding to the output signal Y and the target signal T into a dual-rail signal Z, with the signs paired appropriately, as follows:

1) $C11 + Yp \xrightarrow{k} C11 + Zp$
2) $C11 + Ym \xrightarrow{k} C11 + Zm$
3) $C11 + Tp \xrightarrow{k} C11 + Zm$
4) $C11 + Tm \xrightarrow{k} C11 + Zp.$

Then, drawing on the approach of Buisman *et al.* [21], we set the value of the signal L to be Z^2, using the following three reactions in the main network:

1) $C11 + Zp + Zp \xrightarrow{k} C11 + Zp + Zp + Lp$
2) $C11 + Zm + Zm \xrightarrow{k} C11 + Zm + Zm + Lp$
3) $C11 + Lp \xrightarrow{k} C11.$

Note also that, unlike most species in our circuit design, the concentration of Z will persist after these reactions are finished

executing; we deal with this in the later "reset" phase discussed in Section II-L.

J. Approximating Loss Derivative

To determine the appropriate amount by which to update a particular weight, our learning CRN must approximate the first derivative of the loss with respect to that weight. Here, we describe a version of the learning CRN in which a single weight is updated during each training round.

The fundamental equation to calculate the loss derivative with respect to weight Wi is

$$\frac{\partial L}{\partial Wi} \approx \frac{L(Wi + \Delta Wi) - L(Wi)}{\Delta Wi} \tag{2}$$

which uses quadrature to approximate the derivative by computing the loss when weight Wi is perturbed by ΔWi. The true value of the derivative is approached as ΔWi tends to zero. The exact nature of this dependence will depend on the details of the loss function landscape. The weight update rule is then

$$Wi := Wi - \alpha \times \frac{\partial L}{\partial Wi} \tag{3}$$

which we can rewrite, using (2), as

$$Wi := Wi - \frac{\alpha}{\Delta Wi} \times (L(Wi + \Delta Wi) - L(Wi)). \tag{4}$$

Our learning CRN uses (4) to approximate the loss derivative with respect to a chosen weight by computing the loss in both the main and shadow networks, as described above, and then simply subtracting them and scaling the result. The "learning rate," α, is a small constant factor that limits the rates of change of the weights by scaling our first-order approximation to the true gradient value.

As a practical matter, the value $L(Wi + \Delta Wi) - L(Wi)$ would typically be quite small before it is scaled by $(\alpha/\Delta Wi)$, meaning that it could get lost in the numerical errors introduced by ODE integration. Therefore, we actually multiply the concentration $[Lp]$ (which represents $L(Wi)$, computed by the main network) and the concentration $[Lp']$ (which represents $L(Wi + \Delta Wi)$, computed by the shadow network) by this factor before carrying out the subtraction. The corresponding reactions for scaling the loss from the main network are as follows:

1) $C13 + \text{SCALE} + Lp \xrightarrow{k} C13 + \text{SCALE} + Lp + \text{LScaledp}$
2) $C13 + Lp \xrightarrow{k} C13$
3) $C13 + \text{SCALE} + Lm \xrightarrow{k} C13 + \text{SCALE} + Lm + \text{LScaledm}$
4) $C13 + Lm \xrightarrow{k} C13.$

The concentration of SCALE represents the value of $(\alpha/\Delta Wi)$, which we fix by leaving the learning rate and weight perturbation value unchanged throughout. The value of the weight increment is then calculated according to (4) by the following reactions:

1) $C13 + \text{LScaledp}' \xrightarrow{k} C13 + \text{Deltap}$
2) $C13 + \text{LScaledm}' \xrightarrow{k} C13 + \text{Deltam}$
3) $C13 + \text{LScaledp} \xrightarrow{k} C13 + \text{Deltam}$
4) $C13 + \text{LScaledm} \xrightarrow{k} C13 + \text{Deltap}$

which puts the corresponding value into the dual-rail signal Delta. The signs are flipped in the last two reactions so as to correctly implement the subtraction operation.

K. Weight Updates via Feedback

The last step of the learning process is to update the weights by feeding the value of Delta back to one of the weights. In this version of the learning CRN, we perturb a single weight in each training round. Therefore, only one weight can be updated at a time. To achieve this, in the next clock phase ($C15$) we use a series of reactions catalyzed by "selector" species to route the signal from Delta to increment or decrement the correct weight. This structure is a "demultiplexer" because it redirects one input to one of multiple potential outputs. There are nine weights in our neural network, and therefore, there are nine different selector species: SELECTW00, SELECTW01, ..., SELECTW22. These catalyze the conversion of Delta into the corresponding weight species. In order to keep the weight values stored in the main and shadow networks in sync, there is a fan-out that transfers the value into both the main network and shadow network weight species. Since (3) requires the value of the loss derivative to be subtracted from the corresponding weight, the demultiplexer reactions must also flip the polarity of the dual-rail signal, by translating negative Delta signals into positive weight signals, and vice versa. For example, the reactions for updating weight $W00$ are as follows:

1) $C15 + \text{SELECTW00} + \text{Deltap} \xrightarrow{k} C15 + \text{SELECTW00} + W00m + W00m'$
2) $C15 + \text{SELECTW00} + \text{Deltam} \xrightarrow{k} C15 + \text{SELECTW00} + W00p + W00p'$

and we include similar reactions for all the other eight weights. After this clock phase has ended, both the main and shadow network weights will have been updated by the amount calculated according to the learning rule.

L. Resetting for the Next Training Round

At the end of each round, the system must be reset to leave it in the correct state to initiate the next training round. This takes place in clock phase $C17$. As mentioned above, the dual-rail signal Z that is used to carry out the squaring operation when computing the error will persist after that operation is completed, and therefore we degrade those species in this clock phase

1) $C17 + Zp \xrightarrow{k} C17$
2) $C17 + Zm \xrightarrow{k} C17$

with similar reactions included for the Z signal from the shadow network. We also include similar reactions to degrade the various "selector" species, as otherwise they would persist and we will probably want to perturb a different weight in the next training round.

Finally, as part of the resetting process, a compensatory negative addition of the perturbed shadow weight signal must be added to reset its value to match the corresponding weight in the main network at the end of the training round, so that other weights can subsequently be perturbed.

M. Initial Conditions

The clock signals $C19$ and $C20$ are initially set high, with a concentration of 1.0, and all other clock signals are initially set low, with a concentration of 10^{-6}. The concentration of the species representing the weights are set to the initial values of those weights. It is important that the weight values in the main and shadow networks are both set to the same value initially. Finally, we initialize the concentration of SCALE to the value of the ratio $(\alpha/\Delta Wi)$, as outlined above. In the simulations presented here, we use $\alpha = 1$ and $\Delta Wi = 0.1$, so we initialize SCALE with concentration 10.

N. Training Examples

In the first version of our learning CRN, we perturb one weight at a time. Therefore, each training instance consists of the following: a set of perturbations at the start of the round (i.e., at the start of clock phase $C1$) that provides values for the three input signals via the concentrations of three dual-rail species: BIAS (which is always set to -1), $X1$, and $X2$. The target output value is presented as the concentration of the TARGET species. Depending on the weight that was chosen to be perturbed, an increment of ΔWi is provided by adding more of the shadow network version of that weight. To ensure that the demultiplexer feeds back to the correct weight, a concentration of 1.0 of the correct "selector" signal is provided. Importantly, by queueing up multiple such perturbation sets, we are able to simulate multi-round training of our learning CRN.

O. Single Weight Training Results

We began by implementing the abstract CRN for a circuit that learns by perturbing, and thus modifying, one weight in each training round. Example timecourses of key species in the "main network" over the course of a single training round are presented in Fig. 3. These illustrate the division of reactions between different clock phases, as outlined in Table I. (See Fig. S2 for the corresponding plot for the "shadow network.")

We developed a collection of training schemes that could train the CRN from a single set of initial weights (see Table S1) to implement any one of the 16 two-input Boolean functions. Importantly, because our CRN faithfully implements a mathematically well-defined learning algorithm, we were able to write a "reference" implementation in Python and use this to rapidly screen training procedures (see Methods, Fig. S1, Tables S2 and S3). We used input values of 1 and -1 to represent true and false inputs, respectively, and labeled any output value within 0.1 of the corresponding correct truth value as being correct (see Table S5).

For this learning CRN, the percentage errors in the weights (see Fig. S9) are almost all 0.25% or below, though since only one weight is updated at a time, the error for each weight is precisely zero until it is first updated. To demonstrate that each CRN had learned the corresponding Boolean function, we created an "evaluator" CRN that simply evaluates the neural network in a feedforward manner, without any weight update, and preloaded it with the learned weights corresponding to each of the 16 learned functions. By providing inputs between -1 and 1 for each of the two input signals, we produced heatmaps of the learned decision surface that confirm that they successfully learned to implement the Boolean function when presented with inputs whose values are ± 1. Fig. 4 illustrates example heatmaps for the "X1 NOR X2" and "X1 XOR X2" functions; all 16 heatmaps are presented in Figs. S3–S5. The "XOR" function is of particular interest [23] because it is not linearly separable [24] and therefore requires a multilayer network of the kind constructed here. This demonstrates the correct realization of learning in a multilayer, nonlinear artificial neural network as implemented by our abstract CRN approach.

P. Extension to Perturb All Weights in Parallel

While the version of our CRN that learns by updating one weight at a time can learn binary functions correctly, such a learning algorithm is more prone to getting stuck in local minima and may require a large amount of training rounds to make progress, because updating just one weight means that the system is not moving in the optimal direction in weight space.

We address this potential problem and demonstrate a generalization of our initial approach, by creating an extension of the learning CRN described above that perturbs not just one weight at a time, but all nine. This is essentially an extension of the architecture from Fig. 2 in which there is not just one shadow network but rather nine, each of which perturbs a different weight by the same, fixed, amount. Here, we briefly present the differences and expansions between this version and the single weight version discussed above.

First, as there are nine shadow networks, the inputs must be copied into all nine of these shadow networks, and into the main network. These are named by appending them not just with the prime symbol, but also with a code indicating the weight that is perturbed in that particular shadow network, such as $X00'$, $X01'$, ... , $X22'$. Second, as each of the shadow networks will compute its own loss value, there will be nine different subtraction and scaling operations to carry out. This means that the loss value computed by the main network must be put through another fan-out operation to copy it nine times, so that these distinct species can then be used to compute the weight update values for each of the nine weights. Third, since every weight is updated at each step, some of the machinery for determining which weight is to be perturbed and controlling the feedback loop can be dispensed with. Specifically, each of the shadow networks can be "perturbed" by the addition of a fixed amount of the corresponding weight species in the initial conditions, and by simply updating the weight along with the others, it will remain perturbed by the desired amount relative to the "true" value of that weight as stored in the main network. This means that a weight perturbation does not need to be supplied with every training round and also does not need to be "undone" at the end of the round. It also removes the need for the demultiplexer and the "selector" species, as the feedback can just take each calculated weight update and pass it through a fan-out

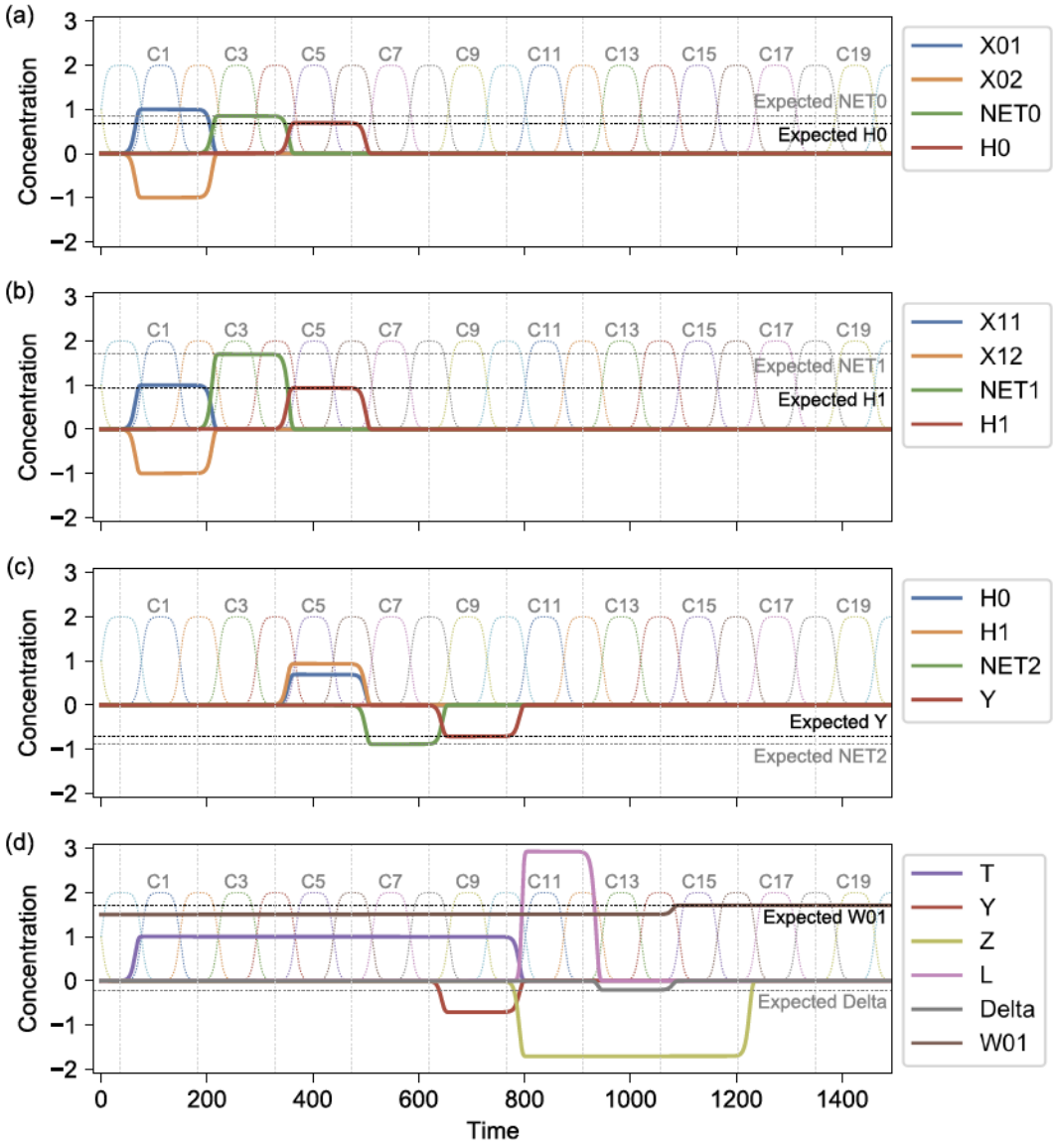

Fig. 3. Time-course of selected dual-rail signals illustrating the operation of the single weight training CRN through a single training round. (a) Input fan-out (clock phase $C1$) and generation of $H0$ output signal from neuron $N0$ (clock phases $C3$ and $C5$). (b) Input fan-out (clock phase $C1$) and generation of $H1$ output signal from neuron $N1$ (clock phases $C3$ and $C5$). (c) Generation of overall Y output signal from neuron $N2$ (clock phases $C7$ and $C9$). (d) Conversion of Y signal into a Delta signal (clock phases $C11$ and $C13$), leading to subsequent modification of the $W01$ weight value via the feedback mechanism (clock phase $C15$).

reaction to copy it to the main network and all of the shadow networks, while inverting the polarity of the dual-rail signal as before.

These changes produce a CRN that is larger in terms of the number of species and reactions that the single weight version outlined above, because of the additional duplicated versions of the neural network. This version of the CRN is substantially larger than the first version (583 species and 1213 reactions, when compared to 144 species and 286 reactions). It also takes longer to simulate (see Table S4). However, as outlined above, it is actually simpler in some respects. This CRN can theoretically learn in fewer training rounds than the single weight version, and furthermore, it should learn more reliably since it follows the optimal path in weight space that leads to the greatest decrease in the computed loss at each step.

Q. Learning Two-Input Binary Functions

Following the approach outlined above, we similarly trained the version of our CRN that perturbs all weights in parallel using a pre-selected set of 10-round training procedures, so it could be trained to learn any of the 16 two-input Boolean functions. The difference in this version was that each weight could potentially be updated in each training round (see Table S6 for results).

As before, we created heatmaps from an evaluator CRN to visualize the learned output surface of the network. Example output heatmaps for the "X1 AND (NOT X2)" and "X1 XNOR

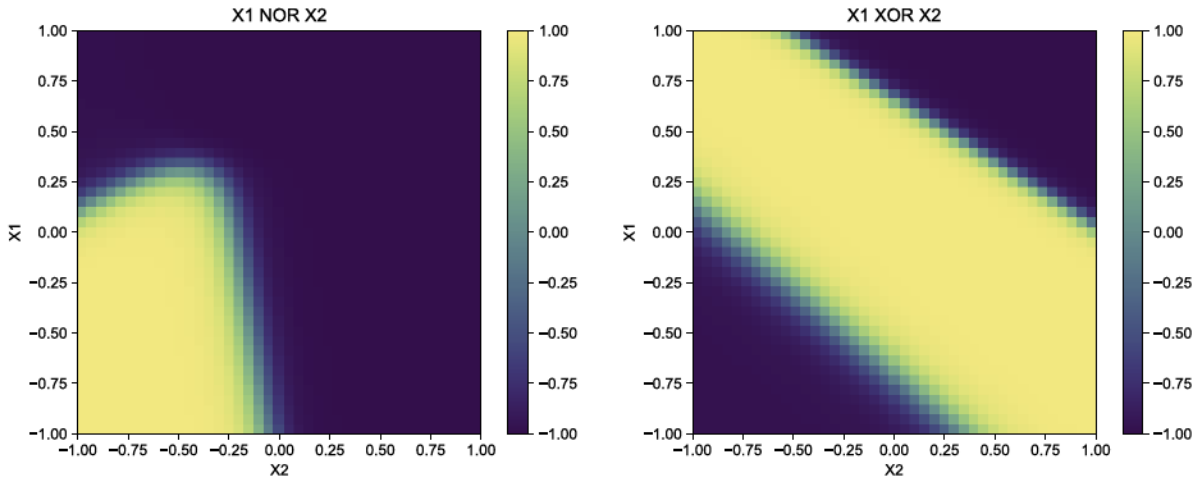

Fig. 4. Example heatmaps showing the learned decision surfaces for selected binary functions: "X1 NOR X2" (left) and "X1 XOR X2" (right). Weights were learned using the CRN that perturbs one weight at a time. Each heatmap was produced from the results of evaluating the learned weights using a non-learning version of the CRN that simply computes the output signal from the network for each of the input combinations.

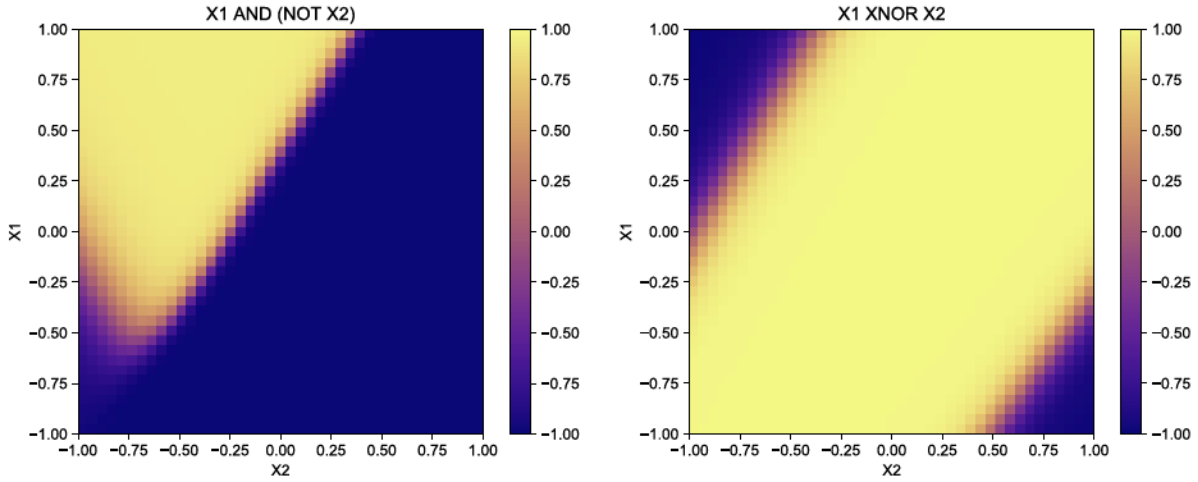

Fig. 5. Example heatmaps showing the learned decision surfaces for selected binary functions: "X1 AND (NOT X2)" (left) and "X1 XNOR X2" (right). Weights were learned using the CRN that perturbs all weights in parallel. Each heatmap was produced from the results of evaluating the learned weights using a non-learning version of the CRN that simply computes the output signal from the network for each of the input combinations.

X2" functions are presented in Fig. 5; all 16 heatmaps are presented in Figs. S6–S8. The "XNOR" function is another example of a function that is not linearly separable, thereby further demonstrating the power of our multilayer neural network. The percentage errors of the weights learned by this CRN over the course of the training procedures relative to the reference implementation (see Fig. S9) are higher than for the first version, because more weights change in each round. In summary, these results demonstrate that we can successfully, and accurately, implement a nonlinear, multilayer training algorithm that is relevant to mainstream machine learning research and can be trained via computationally pre-screened training protocols.

III. Discussion

To summarize, we have presented a design for an abstract chemical reaction network (CRN) that implements a mathematically well-defined training algorithm for a multilayer, nonlinear artificial neural network. The reactions in this CRN scheme are orchestrated by an autonomous molecular clock [19] and we draw upon previous work on implementing certain arithmetic functions in chemical reactions [21]. We have presented a novel mechanism for implementing a nonlinear transfer function (the hyperbolic tangent) in a mathematically accurate manner by exploiting particular properties of its derivatives. We use a non-standard learning algorithm based on direct approximation of partial derivatives of network loss with respect to particular weights that is based on explicit perturbation of those weights [18]. Finally, we demonstrate correct training of these learning networks by training them to implement all 16 of the two-input Boolean functions, both by perturbing a single weight at a time and by perturbing all of the weights in parallel. This includes functions such as "XOR" that are not linearly separable, which provably require a multilayer network to learn correctly. To our knowledge, this work is the first to implement a mathematically well-defined training

algorithm for a multilayer, nonlinear chemical neural network. Our key contribution is the design and implementation of the learning CRN rather than the detail of the learning algorithm itself. Importantly, the experimental realization of the networks proposed here, or similar networks, would lead to the first adaptive and trainable chemical neural networks.

A. Modularity of the Learning Algorithm

A crucial aspect of our network design is that the learning algorithm is agnostic to the details of the network being trained, such as the transfer function used and even the network architecture itself. This is because the learning algorithm simply takes two or more loss values as inputs and uses their difference to compute a loss derivative estimate. This means that the learning algorithm implemented here can be applied unchanged to a wide range of neural networks, thereby providing a general-purpose mechanism for training of chemical neural networks and demonstrating the broad applicability of our approach.

B. Implementation of Nonlinear Transfer Function

An additional contribution to CRN design is our construction for computing the hyperbolic tangent of the weighted sums of the inputs to our neurons. Driven by a desire to implement a continuous, nonlinear, everywhere-differentiable transfer function, we developed a simple reaction motif that precisely computes the value of the hyperbolic tangent. Our approach builds on previous work by Fages *et al.* [22] who used a "change of variables" construction to achieve what we accomplish by just adding an extra reactant as a "counter". To our knowledge, our particular construction is novel. However, it does rely on the ability to express the derivative of the function in terms of itself, and it cannot be applied if the derivative in question is zero when the input is zero. Other approaches to this problem include computing estimates of transcendental functions, including sigmoids, in terms of truncated polynomial series expansions [25]. The advantage of our approach is that it computes the hyperbolic tangent exactly, whereas truncated series expansions are only a reasonable approximation in the vicinity of the value around which they are expanded. Our approach also uses a small number of chemical reactions.

C. Potential Issues With Use of Autonomous Molecular Clock

The different phases of our CRN implementation are orchestrated by the oscillation of an autonomous molecular oscillator which serves as a clock signal, an approach previously explored elsewhere [19], [26]. Some of the reactions, such as those that implement weighting and fan-out, do not necessarily need to be staged in this way. Others, such as the reactions that calculate the hyperbolic tangent transfer function, would seem to require that their inputs have reached a stable state beforehand. The use of such a clock also relies on the deterministic nature of the ODE semantics, because stochastic fluctuations might cause the clock signal to die if any of the clock species fluctuate to zero during a "low" phase. However, the clock species concentrations in the low phases are required to be low to minimize errors due to background activation of reactions out of their specified clock phase.

We used a relatively large number of clock phases in this system in large part to enable easier testing and debugging of the CRN design. However, in principle, fewer clock phases could be used. For example, clock phases $C1$ and $C3$ could be combined into a single phase, as could phases $C5$ and $C7$. Similarly, phases $C9$, $C11$, $C13$, and $C15$ could be combined into one. The initial phase in each set is one where the reactions must wait until their input species are at steady state, for example, the reactions that compute the tanh nonlinearity. Furthermore, in this design, we only used every other clock phase as a catalyst, to reduce overlap in the catalyzed reactions.

However, the use of an autonomous clock signal might not necessarily be required at all for a practical implementation: the corresponding clock species could, at least in principle, be added manually into an open reactor implementation of our scheme, with addition controlled by a computerized microfluidic device. Furthermore, in our simulations, we assume that the addition of chemical species after time $t = 0$ causes no dilution of the reaction volume. This could be approximated by adding small volumes of highly concentrated input, although this would be prone to pipetting error in practice. Future work could assess the effects of dilution on the accuracy of learning circuits such as our overextended rounds of training. Other approaches, such as the use of absence indicators to separate reactions into discrete phases [27], could also be used with our system. The extent to which the clock signal, or other sequencing primitives, is required for correctness of our system, and similar molecular programs, is an interesting question for future research.

D. Learning Performance of the Circuits

The heatmaps of the decision surfaces for our two learning CRNs (Figs. S3–S8) indicate that they learn all 16 two-input Boolean functions correctly when presented with inputs that are exactly 1.0 for "true" and -1.0 for "false," with an allowable error threshold of 0.1. However, the decision surfaces show that the trained networks may not give sensible answers for some intermediate values (e.g., the "X1 NOR X2" heatmap in Fig. S7), because we only trained the networks with input values that were exactly ± 1, to simplify and shorten the training process. Therefore, the algorithm has no need to adjust the decision surface for these intermediate input values. By training the networks with additional intermediate input values, it should be possible to produce decision surfaces that behave more reasonably for intermediate input values. However, this could require longer training procedures. In addition, while we used a fresh "evaluator" network to test the outputs produced by the learning circuits, in principle this could be achieved in a single circuit by the inclusion of a molecular "switch" to disable the learning and feedback components [16].

Furthermore, the errors between our CRN implementation and the reference implementation would compound if longer training sequences were attempted. Some of these errors may be attributed to intrinsic inaccuracy in the numerical

integrator that solves the ODEs, but there are also inherent errors in the CRN computing framework. For example, the arithmetic modules that implement weighting are only accurate at equilibrium, and the molecular clocks do not completely shut off reactions catalyzed by the "low" clock signals, since they are never exactly zero. Reducing such computational errors is likely to be an important area of future research in molecular programming.

To study the extent to which the characteristics of the molecular clock affect circuit performance, we carried out some additional simulations with clocks where the OFF state is less perfect. The effect of this change is to enable reactions to occur at faster rates outside their designated clock phases. To simplify the setup of these simulations, here we used a square wave clock signal generated via model perturbations rather than using an autonomous molecular clock. The results of these simulations are reported in the electronic supplementary material (Figs. S10–S14). These show that the OFF state of the clock can be increased by several orders of magnitude before the results of the learning simulation are affected, thereby demonstrating that our system is relatively robust to the use of an imperfect molecular clock to schedule the reactions.

E. Other Possible Extensions and Improvements

Other possible extensions and improvements to our learning scheme include alternative transfer functions (such as rectified linear units, which are in widespread use in mainstream machine learning research) and different network architectures. Modifying the network architecture would be relatively easy, given the modular nature of our learning algorithm. As we saw when we extended our CRN to perturb all of the weights in the network in parallel, the weight perturbation learning circuit design does not scale gracefully with the size of the network, as the entire network must be duplicated once per weight in that variant of the CRN (this could be viewed as scaling as $O(n_w \times n_n)$, where n_w is the number of weights and n_n is a measure of overall network size). This could be circumvented by implementing an alternative learning algorithm such as backpropagation, which does not require the entire network to be reevaluated for each weight that is to be updated. However, if only one weight is perturbed in each training cycle, the size of the feedback system in our design would remain constant and the enlarged network would only need to be duplicated once (and thus scales as $O(n_n)$ only). This would thus provide better scaling of network size, although more training rounds would be needed to learn the target weights, given that only one weight can be updated per round. In both cases, the number of rate constants required scales as $O(1)$, as we have shown that a fixed number of rate constants suffices for our design. Applying similar techniques to larger neural network architectures would enable more sophisticated behaviors to be learned by our CRN systems.

F. Possible Chemical Implementations

Abstract CRNs, such as those used here, are a powerful programming language for molecular computing systems because they can be compiled into actual chemical implementations. In particular, it has been shown that any abstract CRN can be implemented as a network of DNA strand displacement reactions [8], in which the abstract species from the original CRN are represented by distinct DNA strands and additional DNA complexes mediate transformations between free strands corresponding to the kinetics of the abstract CRN. That result also applies to the higher-order reactions posited here, some of which involve three or more reactants. Thus, our learning circuit design could, at least in principle, be realized as an actual biochemical circuit. It is worth noting, however, that the undesired side reactions known as "leak" would likely reduce the accuracy of a practical DNA strand displacement implementation of our circuit, although recent advances toward gate designs with drastically reduce leak rates [28] could help to mitigate such problems.

Another important issue for practical implementations would be the construction of multiple, parallel, non-interfering "shadow" networks that is required by our construction. In the context of DNA strand displacement reactions, this can be achieved via careful design of the DNA sequences to create distinct species of DNA circuit components that function similarly but do not interact with each other because their nucleotide sequences are sufficiently distinct. Indeed, previous work has used exactly this technique in the context of DNA strand displacement circuits [29]. In that work, the shadow network was created for the purpose of "shadow cancellation," which aims to reduce leak by canceling the output from the main network with that from the shadow network. However, a similar approach could be applied, at least in principle, to create multiple non-interacting shadow networks in a learning CRN implementation.

By way of comparison, the DNA strand displacement seesaw gate circuit for computing the square root of a four-bit binary number that was published in 2011 had "74 initial DNA species, excluding inputs" [2]. The smaller of our two learning circuits has 144 species and 286 reactions, and while these are abstract reactions and not concrete DNA reactions, this could nevertheless soon be within the plausible range if the tools and technologies available to molecular programmers continue to advance at the rapid pace that they have during the last decade. Thus, our work advances the state of the art toward a world in which molecular circuits have non-trivial learning capabilities which can be deployed for scientific as well as practical applications in the real world, such as long-term health monitoring applications for biomedicine.

REFERENCES

[1] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree, "Enzyme-free nucleic acid logic circuits," *Science*, vol. 314, no. 5805, pp. 1585–1588, 2006.

[2] L. Qian and E. Winfree, "Scaling up digital circuit computation with DNA strand displacement cascades," *Science*, vol. 332, pp. 1196–1201, Jun. 2011.

[3] D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree, "Engineering entropy-driven reactions and networks catalyzed by DNA," *Science*, vol. 318, no. 5853, pp. 1121–1125, 2007.

[4] Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig, "Programmable chemical controllers made from DNA," *Nature Nanotechnol.*, vol. 8, pp. 755–762, Sep. 2013.

[5] G. Chatterjee, N. Dalchau, R. A. Muscat, A. Phillips, and G. Seelig, "A spatially localized architecture for fast and modular DNA computing," *Nature Nanotechnol.*, vol. 12, pp. 920–927, Jul. 2017.
[6] N. Srinivas, J. Parkin, G. Seelig, E. Winfree, and D. Soloveichik, "Enzyme-free nucleic acid dynamical systems," *Science*, vol. 358, no. 6369, Dec. 2017, Art. no. eaal2052.
[7] B. Groves *et al.*, "Computing in mammalian cells with nucleic acid strand exchange," *Nature Nanotechnol.*, vol. 11, no. 3, pp. 287–294, Mar. 2016.
[8] D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 12, pp. 5393–5398, Mar. 2010.
[9] D. Y. Zhang and G. Seelig, "Dynamic DNA nanotechnology using strand-displacement reactions," *Nature Chem.*, vol. 3, pp. 103–113, Jan. 2011.
[10] Y. Xiang and Y. Lu, "Using personal glucose meters and functional DNA sensors to quantify a variety of analytical targets," *Nature Chem.*, vol. 3, no. 9, pp. 697–703, Sep. 2011.
[11] C. Jung and A. D. Ellington, "Diagnostic applications of nucleic acid circuits," *Accounts Chem. Res.*, vol. 47, no. 6, pp. 1825–1835, 2014.
[12] L. Qian, E. Winfree, and J. Bruck, "Neural network computation with DNA strand displacement cascades," *Nature*, vol. 475, pp. 368–372, Jul. 2011.
[13] K. M. Cherry and L. Qian, "Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks," *Nature*, vol. 559, pp. 370–376, Jul. 2018.
[14] D. Blount, P. Banda, C. Teuscher, and D. Stefanovic, "Feedforward chemical neural network: An in silico chemical system that learns XOR," *Artif. Life*, vol. 23, no. 3, pp. 295–317, 2017.
[15] P. Banda, C. Teuscher, and M. R. Lakin, "Online learning in a chemical perceptron," *Artif. Life*, vol. 19, no. 2, pp. 195–219, Apr. 2013.
[16] M. R. Lakin, A. Minnich, T. Lane, and D. Stefanovic, "Design of a biochemical circuit motif for learning linear functions," *J. Roy. Soc. Interface*, vol. 11, no. 101, 2014, Art. no. 20140902.
[17] M. R. Lakin and D. Stefanovic, "Supervised learning in adaptive DNA strand displacement networks," *ACS Synth. Biol.*, vol. 5, no. 8, pp. 885–897, Aug. 2016.
[18] M. Jabri and B. Flower, "Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks," *Neural Comput.*, vol. 3, no. 4, pp. 546–565, Dec. 1991.
[19] M. Vasić, D. Soloveichik, and S. Khurshid, "CRN++: Molecular programming language," *Natural Comput.*, vol. 19, no. 2, pp. 391–407, Jun. 2020.
[20] B. Yordanov, J. Kim, R. L. Petersen, A. Shudy, V. V. Kulkarni, and A. Phillips, "Computational design of nucleic acid feedback control circuits," *ACS Synth. Biol.*, vol. 3, no. 8, pp. 600–616, Aug. 2014.
[21] H. J. Buisman, H. M. M. ten Eikelder, P. A. J. Hilbers, and A. M. L. Liekens, "Computing algebraic functions with biochemical reaction networks," *Artif. Life*, vol. 15, no. 1, pp. 5–19, 2009.
[22] F. Fages, G. L. Guludec, O. Bournez, and A. Pouly, "Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs," in *Proc. CMSB*, vol. 10545, J. Feret and H. Koeppl, Eds. Cham, Switzerland: Springer, 2017, pp. 108–127.
[23] R. Bland, "Learning XOR: Exploring the space of a classic problem," Dept. Comput. Sci. Math., Univ. Stirling, Stirling, Scotland, Tech. Rep. 148, 1998.
[24] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, 2nd ed. Cambridge, MA, USA: MIT Press, 1972.
[25] S. A. Salehi, K. K. Parhi, and M. D. Riedel, "Chemical reaction networks for computing polynomials," *ACS Synth. Biol.*, vol. 6, no. 1, pp. 76–83, Jan. 2017.
[26] R. A. Brittain, N. S. Jones, and T. E. Ouldridge, "Biochemical szilard engines for memory-limited inference," *New J. Phys.*, vol. 21, no. 6, Jun. 2019, Art. no. 063022.
[27] H. Jiang, S. A. Salehi, M. D. Riedel, and K. L. Parhi, "Discrete-time signal processing with DNA," *ACS Synth. Biol.*, vol. 2, no. 5, pp. 245–254, 2013.
[28] B. Wang, C. Thachuk, A. D. Ellington, E. Winfree, and D. Soloveichik, "Effective design principles for leakless strand displacement systems," *Proc. Nat. Acad. Sci. USA*, vol. 115, no. 52, pp. E12182–E12191, Dec. 2018.
[29] T. Song *et al.*, "Improving the performance of DNA strand displacement circuits by shadow cancellation," *ACS Nano*, vol. 12, no. 11, pp. 11689–11697, Nov. 2018.

David Arredondo received the B.A. degree in physics from the University of Denver, Denver, CO, USA, in 2014. He is currently pursuing the M.S. degree in computer science and the Ph.D. degree in nanoscience and microsystems engineering with The University of New Mexico, Albuquerque, NM, USA.

His research interests include statistical physics, molecular robotics, and the design of adaptive chemical automata using abstract chemical reaction networks.

Matthew R. Lakin (Member, IEEE) received the Ph.D. degree in computer science from the University of Cambridge, Cambridge, U.K., in 2010.

From 2009 to 2011, he was at Microsoft Research Cambridge, Cambridge, U.K. In 2011, he moved to The University of New Mexico (UNM), Albuquerque, NM, USA, and became a tenure-track Assistant Professor with the UNM Department of Computer Science in 2017. His research interests include DNA nanotechnology, synthetic biology, and biological modeling languages and software tools.