Deep Molecular Programming:
A Natural Implementation of Binary-Weight ReLU Neural Networks

Marko Vasic [1]   **Cameron Chalk** [1]   **Sarfraz Khurshid** [1]   **David Soloveichik** [1]

Abstract

Embedding computation in molecular contexts incompatible with traditional electronics is expected to have wide ranging impact in synthetic biology, medicine, nanofabrication and other fields. A key remaining challenge lies in developing programming paradigms for molecular computation that are well-aligned with the underlying chemical hardware and do not attempt to shoehorn ill-fitting electronics paradigms. We discover a surprisingly tight connection between a popular class of neural networks (binary-weight ReLU aka BinaryConnect) and a class of coupled chemical reactions that are absolutely robust to reaction rates. The robustness of rate-independent chemical computation makes it a promising target for bioengineering implementation. We show how a BinaryConnect neural network trained in silico using well-founded deep learning optimization techniques, can be compiled to an equivalent chemical reaction network, providing a novel molecular programming paradigm. We illustrate such translation on the paradigmatic IRIS and MNIST datasets. Toward intended applications of chemical computation, we further use our method to generate a chemical reaction network that can discriminate between different virus types based on gene expression levels. Our work sets the stage for rich knowledge transfer between neural network and molecular programming communities.

1. Introduction

Although molecular computation cannot compete with electronics based on speed, the goal is to enable computation in contexts where traditional electronics cannot go.

Chemical control modules compatible with the chemical environments within natural or synthetic cells, bioreactors, and in-the-field diagnostics, are all envisioned applications for such technology. Such computation could, for example, recognize disease state based on chemical inputs and actuate drug delivery to the affected cell. The extensive information processing that occurs in natural cellular regulatory networks underlying all complex life, is a strong proof-of-principle that chemical computation is possible and useful.

Networks of coupled chemical reactions (chemical reaction networks, CRNs) are known to be Turing universal (Fages et al., 2017; Soloveichik et al., 2008), but the resulting systematic ways of programming their behavior can result in extremely large reaction networks and exceedingly inefficient computation. Reasoning in chemical reaction space is difficult: even a very small CRN can be hard to analyze. Thus, there were many previous attempts to build intuitive, yet more efficient programming approaches for CRNs. For example, (Senum & Riedel, 2011) describe a number of computational modules including arithmetic modules and control flow. Another example is CRN++ (Vasic et al., 2018), an imperative programming language that compiles to CRNs. Here, we present a different programming paradigm, one that allows translation of neural networks into chemical reactions. In a similar way that writing an image classifier in an imperative language is prohibitively complex, writing a classifier in a molecular programming language such as CRN++ is practically impossible. However, the technique of programming using neural networks opens doors for such applications in the molecular programming community.

We focus on a class of neural networks called binary-weight ReLU neural networks. These networks have binarized weights $\{+1, -1\}$, and were originally popularized due to their computational efficiency in electronics hardware (Courbariaux et al., 2015). We show that binary-weight ReLU networks have a tight correspondence to a class of *rate-independent* CRNs (Chen et al., 2014a;b). In rate-independent CRNs, computation arises solely from the stoichiometric exchange of reactants for products and the equilibrium is independent of reaction rates. The absolute robustness to reaction rates makes rate-independent CRNs

[1]The University of Texas at Austin, USA. Correspondence to: Marko Vasic <vasic@utexas.edu>.

a promising implementation target for bioengineering.

We first demonstrate our approach by training classifiers on the widely used machine learning datasets IRIS and MNIST. A promising envisioned application of molecular computation lies in medical diagnostics and so called "smart drugs" which activate in response to specific molecular cues. Thus, we next train a classifier to differentiate between four viral infections using chemical information as input (gene expression levels). In all cases, we translate neural networks into CRNs, and simulate CRN behavior in a chemical kinetics simulation framework. The chemical reaction networks exhibit the same output as their corresponding neural networks.

Section 2 first reviews CRNs, and how we can view their behavior as computation, then reviews binary-weight ReLU networks and the BinaryConnect (Courbariaux et al., 2015) method for training them. In Section 3, we first introduce a simple set of chemical reactions which compute the ReLU function, then describe our general technique to compile binary-weight ReLU networks into CRNs which compute the same output. We also describe an optimization procedure which reduces the number of chemical reactions required to implement the neural network. In Section 4, we give simulation results on our chemical classifiers for IRIS, MNIST, and viral infection classification, and verify that their outputs match the neural networks they implement.

2. Background

In this section we provide a brief description of chemical reaction networks and BinaryConnect neural networks.

2.1. Chemical Reaction Networks

Chemical reaction networks (CRNs) formally model the time evolution of concentrations of chemical species in a solution undergoing chemical change. CRNs are typically used to understand naturally occurring chemistry and abstract the behavior of existing biological regulatory networks. The CRN formalism thus provides a standardized way to specify the interaction rules that we expect interacting chemicals to obey in order to perform computation. Closely related models from distributed computing include population protocols (Angluin et al., 2006), Petri nets (Petri, 1966), and vector addition systems (Karp & Miller, 1969).

A CRN is formally a set of chemical *species* (typically written with capital letters) and a set of *reactions* between them. A *state* of the CRN is an assignment of a nonnegative real value to each species, representing the species' concentrations (quantity per unit volume). As the state varies over time, we denote by $a(t)$ the concentration of species A at time t. Each reaction is composed of two multisets

over the species, the *reactants* and *products*, which specify the stoichiometry (the relative quantities of the involved species). For example, species A and B might react to produce species C, which we write as the reaction

$$A + B \longrightarrow C.$$
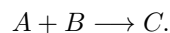
Each reaction has an associated *rate constant* which affects the dynamics of state change of the system. Under typical mass-action kinetics, the instantaneous rate of a reaction is the product of the rate constant and the concentrations of the reactants (i.e., $k \cdot a(t) \cdot b(t)$ for the reaction above).[1] The behavior of the whole system is then described by a set of ordinary differential equations (ODEs) generated based on the reactions. For example, if the reaction above is the only one producing C, the kinetics of C is governed by $dc(t)/dt = k \cdot a(t) \cdot b(t)$.

To define CRN computation, some chemical species are considered *input species*, and some others, disjoint from the first, are considered *output species*. The initial concentrations of the input species are the input to the computation. The system asymptotically converges to a state where the concentrations of the output species are equal to the desired output values.

Many CRN computations work under strong assumptions about the rates of the reactions, but engineering reactions with precise rates is difficult, and small changes in the environment can further disturb these rates. Instead, computation can be achieved by stoichiometry alone (Chen et al., 2014a;b), which forms the basis of the computation considered in this paper. The essential example is the reaction we have already seen: $A + B \longrightarrow C$. The concentration of C as time approaches infinity is the $\min$ of the initial concentrations of A and B, since the reaction can occur only as long as both A and B are present. This fact holds for any (nonzero) reaction rate. Thus we say this CRN rate-independently computes the $\min$ function: $c = \min(a(0), b(0))$.

A programmable chemical process called DNA strand displacement can in principle (and, to some extent, experimentally (Chen et al., 2013; Srinivas et al., 2017)) implement arbitrary, rationally designed CRNs (Soloveichik et al., 2010). Watson-Crick base pairing enables programmability, since interactions are mediated by nucleotide sequence complementarity. In the basic DNA strand displacement reaction, an "output" strand is initially bound to a complementary strand. A single-stranded "in-

---

[1] Although the details of CRN kinetics is not essential for understanding the results in this paper, the numerical simulations are done using continuous mass-action kinetics. In other models of chemical kinetics the amounts of species are represented by nonnegative integer molecular counts and the system corresponds to a continuous time Markov chain. Our results apply to this setting as well, as long as real-valued signals are properly discretized.

put" binds to the double-stranded complex, displacing the "output" strand. Extended designs based on this elementary reaction allow enzyme-free synthetic DNA systems to behave as arbitrary CRNs.[2]

Rate-independent computation aligns well with the design of DNA strand displacement systems. The stoichiometry of a strand displacement reaction is engineered by setting regions on two or more strands to be complementary or orthogonal. In contrast, setting precise reaction rates requires designing specific sequences that achieve desired hybridization free energies. Hybridization rates are highly dependent on the temperature and on the chemical composition of the solution. Further, for a particular chemical implementation, there is some maximum rate possible, so requiring some reactions to be fast and others slow necessarily requires slowing the entire system down.

Although concentrations of species are nonnegative, to imitate the computation done by a neural network we need to store and process negative values. To do so, we represent negative values by the *dual-rail* convention: a value x is represented not by the concentration of one species X, but by the difference in concentration between two species X^+ and X^- (Chen et al., 2014b). More precisely, at time t, $x(t) = x^+(t) - x^-(t)$. Note that values have multiple representations: $x = 3$ can be represented as either $x^+ = 3$ and $x^- = 0$ or $x^+ = 4$ and $x^- = 1$. Although the value is the same, the choice of representation affects convergence time. (Convergence time is an important question for future work as mentioned in Section 6.)

2.2. Binary-Weight ReLU (BinaryConnect)

In recent years we have seen a growing interest in building specialized deep learning hardware to enable new and more complex deep learning applications as well as deploying deep learning systems on low-power devices (Chen et al., 2014c; Dean et al., 2012; Kim et al., 2009). One direction of work tries to tackle this problem by designing more computationally efficient neural networks (Courbariaux et al., 2015; Hubara et al., 2016; Li et al., 2016; Simons & Lee, 2019). One of the initial works in this area introduces BinaryConnect networks (Courbariaux et al., 2015) which restrict the weights of neural networks to values ± 1. This restriction on the weight values enables replacement of most of the multiply units by simple accumulators which leads to a drastic reduction of computational expenses, considering

_____

[2]No reaction is truly irreversible; that is to say, if $A + B \longrightarrow C$, then $C \longrightarrow A + B$ with some non-zero rate. However, nature provides many examples of reactions which are effectively irreversible due to a large change in free energy between reactions and products. Reactions implemented by DNA strand displacement are thermodynamically driven by the formation of additional nucleotide base pairs, which provide enough free energy for the reaction to behave effectively irreversibly.

that the multipliers are the most space and power hungry components of specialized deep learning hardware. Moreover, in spite of severe restriction on the weight values, BinaryConnect still achieves near state-of-the-art results on standard machine learning datasets.

BinaryConnect networks are trained similarly to traditional neural networks through a repeated sequence of: (a) forward-propagation pass computing the output of a network given an input; (b) backward-propagation pass computing error terms (derivatives of the cost function in respect to weights and bias terms); and (c) parameter updates (updating weight and bias terms based on the error terms). BinaryConnect maintains real-valued weights throughout training, but discretizes them to 1 or -1 (if positive or negative, respectively) during the forward and backward-propagation passes. The parameter update is then performed on the real-valued weights. The final binary-weight neural network is generated by discretizing the resulting real-valued weights at the end of training. (Other variations of BinaryConnect training, such as stochastic discretization, are described in the original paper.)

3. Technique

First we describe the technique for compiling binary-weight ReLU networks to CRNs. Then we show how to optimize the CRNs to reduce the total number of reactions. Ultimately, there will be one reaction per ReLU node.

3.1. ReLU Chemical Reaction Network

We propose the following CRN to compute the ReLU function:

$$X^+ \longrightarrow M + Y^+ \tag{1}$$
$$M + X^- \longrightarrow Y^- \tag{2}$$
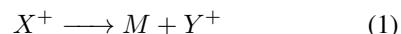
In order to understand why the above reactions compute ReLU, suppose we start with $x^+(0)$ amount of X^+ and $x^-(0)$ amount of X^- and no other species. These concentrations represent the input value $x(0) = x^+(0) - x^-(0)$ in dual-rail form. Although the two reactions will be happening in parallel, the first reaction converges to producing $x^+(0)$ amount of M and Y^+. Therefore, the second reaction converges to producing $\min(x^+(0), x^-(0))$ of Y^-. This implies that the system converges to the dual-rail value of the output $y = y^+(t) - y^-(t) = x^+(0) - \min(x^+(0), x^-(0)) = (x^+(0) - x^-(0)) - \min(x^+(0) - x^-(0), x^-(0) - x^-(0)) = x - \min(x, 0) = \max(x, 0)$. Note that this CRN is rate-independent because the computation will be correct no matter what the rates of the individual reactions are.

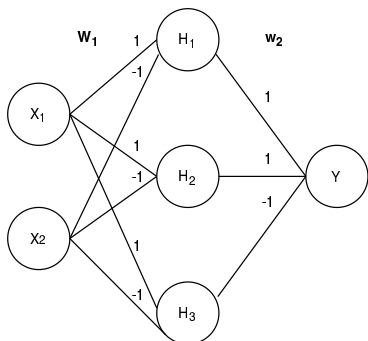Since the output species Y^+ and Y^- are not consumed in the above CRN, they can be used as input species for
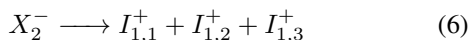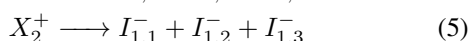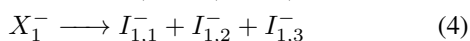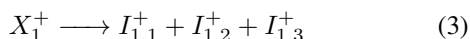
Figure 1. Binary-weight neural network.

any composed downstream computation without interfering with the computation of ReLU. Reactions 1–2 form the simplest CRN computing ReLU in the sense that there is no composable CRN with fewer than 2 reactions or fewer than 5 species computing this function (Vasic et al., 2019).[3]

3.2. Compiling Binary-Weight ReLU Networks

Figure 1 shows an example binary-weight ReLU network. This network consists of an input layer, a single hidden layer with the ReLU activation function, and an output layer without an activation function. Let us initially ignore the bias terms. Then the output of the network is defined by $y = ReLU(\boldsymbol{x}^\top \cdot \boldsymbol{W_1}) \cdot \boldsymbol{w_2}$, where $\boldsymbol{x} \in \mathbb{R}^2$ is an input vector, $\boldsymbol{W_1} \in \{-1,1\}^{2\times 3}$ is a weight matrix into the hidden layer, $\boldsymbol{w_2} \in \{-1,1\}^3$ is a weight vector into the output layer, and $y \in \mathbb{R}$ is the output. The Figure 1 neural network weight values are:

$$\boldsymbol{W_1} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix}, \boldsymbol{w_2} = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}^\top$$
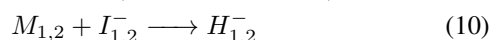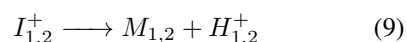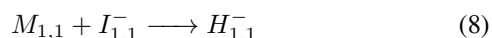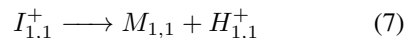
We now convert this binary-weight ReLU network into a CRN. Initially, for each input x_i in the input vector, species X_i^+ and X_i^- will be present such that $x_i^+(0) - x_i^-(0) = x_i$. We multiply the input vector by the weights $\boldsymbol{W_1}$ by the following reactions:

$$X_1^+ \longrightarrow I_{1,1}^+ + I_{1,2}^+ + I_{1,3}^+ \tag{3}$$
$$X_1^- \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \tag{4}$$
$$X_2^+ \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \tag{5}$$
$$X_2^- \longrightarrow I_{1,1}^+ + I_{1,2}^+ + I_{1,3}^+ \tag{6}$$

Species $I_{l,i}$ represents the ith intermediate (before applying nonlinearity) species of the layer l. For a weight with value

---

[3]There is a simpler, non-composable ReLU computing CRN: $X^+ \longrightarrow Y^+$, $Y^+ + X^- \longrightarrow W$. ($W$ is an inert waste species.) It cannot be combined with another CRN which uses species Y^+ as an input, as Y^+ may be consumed by the downstream CRN before it is annihilated by the second reaction. Composability is a well-understood feature in rate-independent CRNs (Chalk et al., 2018; Severson et al., 2019).

1 we include a reaction with a positive input species as a reactant and positive output species as a product, as that has the effect of addition to the product species' value. For a weight with value -1 we include a reaction with a positive input species as a reactant and negative output species as a product, as that has the effect of subtraction from the product species' value. To maintain the dual-rail convention we also include a reaction which contains all the same species with signs flipped. Next, to implement the ReLU activation function we use the module discussed above (reactions 1–2), with the appropriate renaming of species:

$$I_{1,1}^+ \longrightarrow M_{1,1} + H_{1,1}^+ \tag{7}$$
$$M_{1,1} + I_{1,1}^- \longrightarrow H_{1,1}^- \tag{8}$$

$$I_{1,2}^+ \longrightarrow M_{1,2} + H_{1,2}^+ \tag{9}$$
$$M_{1,2} + I_{1,2}^- \longrightarrow H_{1,2}^- \tag{10}$$

$$I_{1,3}^+ \longrightarrow M_{1,3} + H_{1,3}^+ \tag{11}$$
$$M_{1,3} + I_{1,3}^- \longrightarrow H_{1,3}^- \tag{12}$$
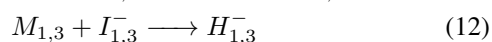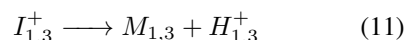
Each intermediate species $I_{l,i}$ of the hidden layer l is a part of a separate ReLU module producing output species $H_{l,i}$. Finally, to multiply by the weight vector $\boldsymbol{w_2}$ we use the following reactions (similar to the multiplication by $\boldsymbol{W_1}$):

$$H_{1,1}^+ \longrightarrow Y^+ \tag{13}$$
$$H_{1,1}^- \longrightarrow Y^- \tag{14}$$
$$H_{1,2}^+ \longrightarrow Y^+ \tag{15}$$
$$H_{1,2}^- \longrightarrow Y^- \tag{16}$$
$$H_{1,3}^+ \longrightarrow Y^- \tag{17}$$
$$H_{1,3}^- \longrightarrow Y^+ \tag{18}$$

The Y species are the output species, and the value $y = y^+(t) - y^-(t)$ approaches $ReLU(\boldsymbol{x}^\top \cdot \boldsymbol{W_1}) \cdot \boldsymbol{w_2}$ as t goes to infinity.

The complete equation of the neural network with bias terms is: $y = ReLU(\boldsymbol{x}^\top \cdot \boldsymbol{W_1} + \boldsymbol{b_1}^\top) \cdot \boldsymbol{w_2} + b_2$. To encode bias terms $\boldsymbol{b_1}$, it is enough to set initial concentrations of the hidden layer $I_{1,i}$ species accordingly (e.g., $i_{11}^+(0) - i_{11}^-(0)$ should be equal to the first component of vector $\boldsymbol{b_1}$); similarly for b_2 and Y species.

We summarize the compilation procedure in Algorithm 1. The input of the algorithm is a binary-weight ReLU network (nn), and the output is its CRN equivalent. We iterate through all the layers of nn starting from the first hidden layer and including the output layer (layer number 1 is the input layer). If a layer does not contain the nonlinear activation (ReLU) we include reactions directly producing the layer output species (named H), otherwise intermediate

Algorithm 1 NNCompile(± 1-weight neural network: nn)

1: $crn = newCRN()$
2: **for** $l = 2$ **to** $nn.numLayers()$ **do**
3:    $layer = nn.getLayer(l)$
4:    $W, b, a = layer.params()$
5:    $pName = $ 'H' **if** $a.linear()$ **else** 'I'
6:    **for** $i = 1$ **to** $W.dimsX$ **do**
7:       $rxn = newReaction()$
8:       $rxn.reactants = \{X_i^+ $ **if** $l == 1$ **else** $H_{l-1,i}^+\}$
9:       **for** $j = 1$ **to** $W.dimsY$ **do**
10:          $pSign = $ '$+$' **if** $W[i,j] == 1$ **else** '$-$'
11:          $rxn.products.add(pName_{l,j}^{pSign})$
12:       **end for**
13:       $crn.add(rxn)$
14:       $crn.add(rxn.reverseSigns())$
15:    **end for**
16:    **for** $j = 1$ **to** $W.dimsY$ **do**
17:       **if** $b[j] > 0$ **then**
18:          $crn.setConc(pName_{l,j}^+, b[j])$
19:       **else**
20:          $crn.setConc(pName_{l,j}^-, -b[j])$
21:       **end if**
22:       **if** $a.nonlinear()$ **then**
23:          $crn.add(I_{l,j}^+ \longrightarrow M_{l,j} + H_{l,j}^+)$
24:          $crn.add(M_{l,j} + I_{l,j}^- \longrightarrow H_{l,j}^-)$
25:       **end if**
26:    **end for**
27: **end for**
28: **return** crn

species are produced first (named I); the name of the appropriate product species is stored in $pName$ (line 5). We iterate through the rows of the weight matrix (W), creating one reaction per row, where the reactant is $H_{l-1,i}^+$ (ith unit of the previous layer), or X_i^+ in the case where the previous layer is the input layer (line 8). To construct the products of the reaction, we iterate through all of the columns of W, adding $pName$ species with positive sign when the weight is $+1$ or negative sign when the weight is -1 (lines 10–11). For a valid dual-rail computation, we include such a constructed reaction to the CRN (line 13), as well as the same reaction with the signs of all the species flipped (line 14). Next, we set the initial concentrations of $pName$ species to correspond to the bias terms. If a bias term is positive, we set the positive species to $b[j]$, otherwise we set the negative species to $-b[j]$ (lines 17–21). Finally, if the layer contains a ReLU activation, we include the ReLU CRN taking as inputs layer intermediate (I) species and producing layer output (H) species. Note that unlike in the previous example, the network output species will be denoted by $H_{N,j}$ (where N is the number of layers in the network).
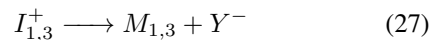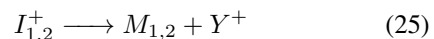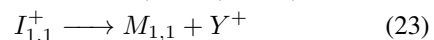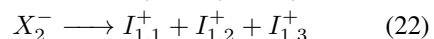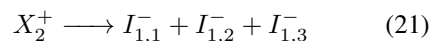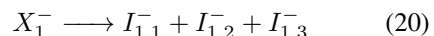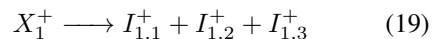
To compute with the CRN, we provide input x by setting the initial concentrations of the input species X as follows. If a component x_i of the input vector is non-negative, then the concentration of the positive input species X_i^+ is set to x_i; otherwise the concentration of the negative input

species X_i^- is set to $-x_i$. Note that in chemistry, reactions occur in parallel, so layers begin their computation before the previous ones complete, unlike typical neural network architectures.
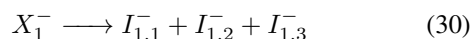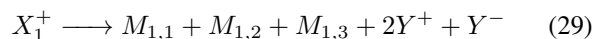
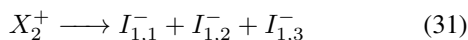### 3.3. Optimization: Reducing the Compiled CRN

We find that unimolecular reactions, such as the first reactions of ReLU modules, can be eliminated from the CRN by altering the bimolecular reactions and the initial concentrations of the CRN species, a process which we describe next. Unimolecular reactions are those with exactly one reactant like $A \longrightarrow B + C$. Whenever A is produced in another reaction, we can replace it with $B + C$. For example, if there is another reaction $X \longrightarrow A + B$, we replace the reaction with $X \longrightarrow 2B + C$. Further, we adjust the initial concentrations of the product species (B and C) by increasing them by the initial concentrations of the reactant (A). Importantly, this transformation works only if A is not a reactant in any other reaction; for example, if there were another reaction like $X + A \longrightarrow Y$, it is not clear what to replace instances of A with, and indeed it is not possible to remove the unimolecular reaction in that case. Luckily, our construction has the property that any species occurs as a reactant in at most one reaction.

We will now illustrate the aforementioned elimination procedure on the neural network CRN presented in Section 3.2, characterized by reactions 3–18. We start by removing the downstream-most unimolecular reactions (reactions 13–18), although the order does not matter. Removing these reactions and adjusting the products accordingly results in the following CRN:

$$X_1^+ \longrightarrow I_{1,1}^+ + I_{1,2}^+ + I_{1,3}^+ \tag{19}$$
$$X_1^- \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \tag{20}$$
$$X_2^+ \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \tag{21}$$
$$X_2^- \longrightarrow I_{1,1}^+ + I_{1,2}^+ + I_{1,3}^+ \tag{22}$$
$$I_{1,1}^+ \longrightarrow M_{1,1} + Y^+ \tag{23}$$
$$M_{1,1} + I_{1,1}^- \longrightarrow Y^- \tag{24}$$
$$I_{1,2}^+ \longrightarrow M_{1,2} + Y^+ \tag{25}$$
$$M_{1,2} + I_{1,2}^- \longrightarrow Y^- \tag{26}$$
$$I_{1,3}^+ \longrightarrow M_{1,3} + Y^- \tag{27}$$
$$M_{1,3} + I_{1,3}^- \longrightarrow Y^+ \tag{28}$$

Note that the initial concentrations of Y species are unaffected as species $H_{1,i}$ are initially set to 0. Next, we remove the now downstream-most unimolecular reactions (reactions 23,25,27) and obtain the following CRN:

$$X_1^+ \longrightarrow M_{1,1} + M_{1,2} + M_{1,3} + 2Y^+ + Y^- \tag{29}$$
$$X_1^- \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \tag{30}$$

$$X_2^+ \longrightarrow I_{1,1}^- + I_{1,2}^- + I_{1,3}^- \qquad (31)$$

$$X_2^- \longrightarrow M_{1,1} + M_{1,2} + M_{1,3} + 2Y^+ + Y^- \qquad (32)$$

$$M_{1,1} + I_{1,1}^- \longrightarrow Y^- \qquad (33)$$

$$M_{1,2} + I_{1,2}^- \longrightarrow Y^- \qquad (34)$$

$$M_{1,3} + I_{1,3}^- \longrightarrow Y^+ \qquad (35)$$

The initial concentrations of $M_{1,1}$, $M_{1,2}$, $M_{1,3}$, Y^+ and Y^- are also affected:

$$m_{1,i}(0) \mathrel{+}= i_{1,i}^+(0), \forall i \in \{1,2,3\}$$
$$y^+(0) \mathrel{+}= i_{1,1}^+(0) + i_{1,2}^+(0)$$
$$y^-(0) \mathrel{+}= i_{1,3}^-(0)$$

The only remaining unimolecular reactions are those whose reactants are input species (X). We do not apply the above elimination procedure to these unimolecular reactions because doing so would remove the input species entirely. Then giving the input to the CRN would require setting the initial concentrations of a number of other species, complicating the input representation.

Note that the CRN can be further simplified while preserving dual-rail values. Reactions 29 and 32 can be reduced to:

$$X_1^+ \longrightarrow M_{1,1} + M_{1,2} + M_{1,3} + Y^+ \qquad (36)$$

$$X_2^- \longrightarrow M_{1,1} + M_{1,2} + M_{1,3} + Y^+ \qquad (37)$$
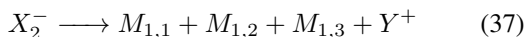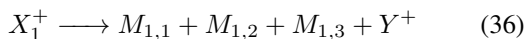
Further, the initial concentrations of Y^+ and Y^- can be reduced by the same amount to make the smaller one zero.

In general, the CRN resulting from the optimization procedure has the following properties. There are no unimolecular reactions besides the input layer, for which there are two reactions per input. There is one bimolecular reaction per ReLU node of the original network. There are no other reactions. Optimization of some adversarial ReLU networks results in reactions with a number of products exponential in the depth of the network. Understanding the scaling of the number of products is an important avenue for future work to ensure feasible CRNs.

Algorithm 2 summarizes the reduction procedure. The algorithm first collects a list of all unimolecular reactions (line 1). Then, for each unimolecular reaction ($R \longrightarrow \sum_i P_i$) the following steps are done: (a) if reactant R is an input species the following steps are skipped and the next reaction is processed (line 4); (b) for every reaction in the CRN producing R, the list of products is altered by removing the product R and adding products P_i (lines 6–11); (c) the initial concentrations of species P_i are increased by the amount of the initial concentration of R (line 13); and (d) the unimolecular reaction is removed from the CRN (line 15). After these steps we perform the following behavior-preserving transformations that further simplify the CRN. For each dual-rail value represented by S^+

Algorithm 2 reduce(CRN: crn)

```
 1: uniRxns = crn.getUnimolecularReactions()
 2: for all uniRxn ∈ uniRxns do
 3:     R = uniRxn.getFirstReactant()
 4:     if crn.inputSpecies(R) then continue
 5:     r(0) = crn.getConc(R)
 6:     for all rxn_i ∈ crn.reactions do
 7:         if rxn_i.containsProduct(R) then
 8:             rxn_i.removeProduct(R)
 9:             rxn_i.addProducts(uniRxn.products)
10:         end if
11:     end for
12:     for all P_i ∈ uniRxn.products do
13:         crn.addConc(P_i, r(0))
14:     end for
15:     crn.remove(uniRxn)
16: end for
17: for all S^+, S^- ∈ crn.getDualRailPairs() do
18:     for all rxn_i ∈ crn.reactions do
19:         while S^+, S^- ∈ rxn_i.products do
20:             rxn.removeProduct(S^+)
21:             rxn.removeProduct(S^-)
22:         end while
23:     end for
24:     m = min(crn.getConc(S^+), crn.getConc(S^-))
25:     crn.addConc(S^+, -m)
26:     crn.addConc(S^-, -m)
27: end for
```

---

and S^- species we: (i) for every reaction whose products multiset contain both S^+ and S^-, remove both species as long as both are present (lines 18–23); (ii) reduce the initial concentrations of S^+ and S^- by the minimum of their values (lines 24–26).

4. Experiments

In this section we describe computational experiments showcasing compilation from neural networks to CRNs. We train BinaryConnect networks on IRIS (Anderson, 1936; Fisher, 1936), MNIST (LeCun et al., 1998), and virus infection datasets (GSE73072). We then translate trained neural networks to CRNs following our compilation technique (Section 3), and simulate the reactions' behavior using an ODE simulator (CRNSimulator). The simulator models CRNs with real-valued concentrations in the standard mass-action model of chemical kinetics, where the rate of a reaction is directly proportional to the concentrations of the reactants. Our main goal is to show the equivalence of the trained neural network and the compiled CRN, and not to improve accuracy of machine learning models, which is orthogonal to our work.

4.1. Datasets

IRIS. The IRIS dataset consists of 150 examples of 3 classes of flowers (Setosa, Versicolor or Virginica), and

4 features per example (sepal length and width, and petal length and width). Considering the small dataset size and that our goal is to show the equivalence of the neural network and the translated CRN, we use the whole dataset for both training and evaluation.

MNIST. The MNIST dataset consists of labeled handwritten digits, where features are image pixels, and labels are digits (0 to 9). We split the original MNIST training set consisting of $60,000$ images into $50,000$ for the training set, and $10,000$ for the validation set. We use the original test set consisting of $10,000$ images. In preprocessing, we center the images (as is done in the BinaryConnect work), and additionally we scale them from 28×28 to 14×14.

Virus Infection. For the virus infection classifier, we used data from NCBI GSE73072 (GSE73072). The dataset contains microarray data capturing gene expression profiles of humans, with the goal of studying four viral infections: H1N1, H3N2, RSV, and HRV (labels). There are 148 patients in the dataset, each with about 20 separate profiles taken at different times during their infection period, for a total of $2,886$ samples. The dataset contains information about which patient was infected and during which point of time. We filter the samples leaving only those that correspond to an active infection, and thus make the data suitable for classification of the four viruses. Finally, we have a total of 698 examples, split into 558 for training, 34 for validation, and 104 for testing. Each sample measures expression of $12,023$ different genes (features); we use the 10 most relevant genes as features which are selected using GEO2R tool (GEO2R) from the NCBI GEO.

4.2. Results

IRIS. We train a neural network with a single hidden layer consisting of 8 units, 4 input units (capturing the features of IRIS flowers), and 3 output units where the unit with the highest value determines the output class. Given that our primary goal is to show the equivalence of a neural network and the compiled CRN, and since the dataset size is quite small (150 examples), we train and evaluate on the whole IRIS dataset. We achieve accuracy of 94% (141 out of 150 examples correctly classified) with a trained BinaryConnect neural network. The equivalent CRN consists of 40 chemical reactions (unoptimized compilation), and 16 chemical reactions (optimized compilation). We simulate both versions of the CRN and confirm that their outputs (labels) match the neural network in all 150 examples. Figure 2 shows the simulation results of the IRIS CRNs on an example input, which is classified with label 0 (since the y_0 output is largest). In both CRNs, the final simulation values of y_0, y_1 and y_2 match the output units values of the neural network (approaching up to 4 decimal places in the time of simulation). Thus not only does the CRN have the

same classification output, but also the numerical values of the individual dual-rail outputs match the output units of the neural network. In regard to the dynamics of CRN convergence, the reduced CRN converges significantly faster.

MNIST. We train a binary-weight neural network with 2 hidden layers and 512 units per each hidden layer, where hyperparameter values are obtained via a random search. We downscale MNIST images from original 28×28 resolution to 14×14; thus the neural network has 14^2 input units (one per pixel). We use 10 output units (for digits 0 to 9), and train the neural network to maximize the output unit corresponding to the correct digit. Our trained model achieves accuracy of 97.76% on the test set. Note that we did not focus on achieving high accuracy; BinaryConnect in the original paper achieves accuracy of over 98%, but uses larger networks (3 hidden layers with 1024 units each). We translate the network to an equivalent CRN consisting of 4488 chemical reactions (unoptimized compilation), and 1024 chemical reactions (optimized compilation). The CRN consists of $2 \cdot 14^2$ input species (two species per input unit encoding positive and negative parts), and similarly $2 \cdot 10$ output species. We simulate the CRN on 100 randomly chosen examples from the test set, and confirm that the output matches that of the neural network in all of the cases. Figure 3 shows simulation results on an example input image (unoptimized CRN). In regard to the dynamics of convergence, note the complex non-monotonic behavior resulting from the mass-action kinetics.

Virus Infection. We train a neural network with 1 hidden layer with 32 units, 10 input units capturing expression of different genes, and 4 output units classifying between virus infections. We achieve test set accuracy of 95.20%. The equivalent CRN consists of 148 chemical reactions (unoptimized compilation), and 32 chemical reactions (optimized compilation). We simulate the CRN on 100 randomly chosen examples from the test set and confirm that the output matches that of the neural network in all cases. Figure 4 shows simulation of the unoptimized CRN on an example that is misclassified.

The performed experiments empirically confirm the soundness of our compilation technique.

4.3. Training Specifics

We use the published implementation of BinaryConnect networks (Courbariaux et al., 2015), and follow the training procedure outlined in Section 2.2. We do not use Batch Normalization (Ioffe & Szegedy, 2015) because it would incur multiplication and division operations at the inference stage (training stage is not a problem) that would be hard to efficiently implement in CRNs. Instead, we rely on Dropout (Srivastava et al., 2014) (stochastically dropping out units in a neural network during training) as a regular-
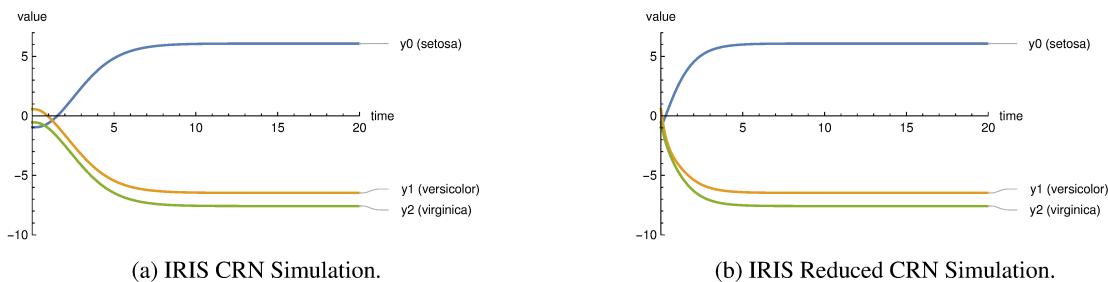
(a) IRIS CRN Simulation.

(b) IRIS Reduced CRN Simulation.

Figure 2. Simulation of IRIS CRN (left) and IRIS CRN in reduced form (right) on the same example. Plots show concentrations of the output species over time, i.e., the difference between positive and negative output species ($y_i(t) = y_i^+(t) - y_i^-(t)$). Networks classify given input with label 0. The reduced CRN converges significantly faster. Note that the initial concentrations of output species change after optimization based on the biases in the ReLU network, as described in Section 3.3.
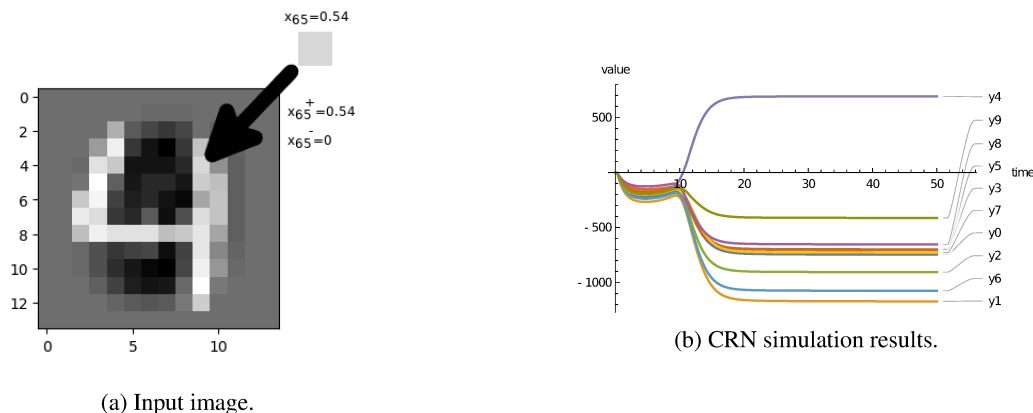

(a) Input image.

(b) CRN simulation results.

Figure 3. Simulation of the MNIST CRN (part b) on an input image (part a). The image is encoded via the input species concentrations: e.g., value 0.54 of the 65th pixel (4th row, 9th column) is represented via initial concentrations of x_{65}^+ and x_{65}^- species. Simulation shows concentrations of the output species over time, i.e., the difference between positive and negative output species ($y_i = y_i^+ - y_i^-$). The CRN classifies the input image as a number 4 (as y_4 has the highest value).
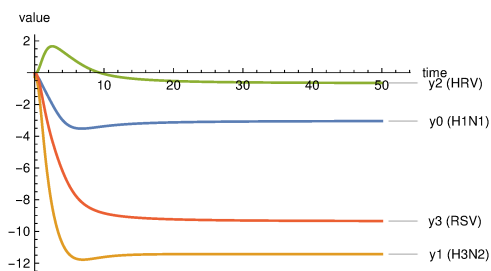

Figure 4. Simulation of Virus Infection CRN on an example (misclassified example). Example is classified as HRV virus, while the true label is H1N1.

ization technique. In all our experiments we use the square hinge loss (as in BinaryConnect) with ADAM optimizer.

We train on MNIST dataset for 250 epochs, measuring the validation accuracy at each epoch, and returning the model that achieves the best validation accuracy during training. We train on IRIS dataset for 10,000 epochs, and return the best performing epoch. We train on the Virus Infection

dataset for 200 epochs, and return the model that achieved the best validation set accuracy. We use an exponentially decaying learning rate.

The rate constants of all reactions are set to 1, and all chemical simulations are performed for 50 arbitrary time units in the CRNSimulator package (CRNSimulator).

5. Related Work

It has long been observed that biological regulatory networks arguably behave in manner analogous to neural networks. For example, both phosphorylation protein-protein interactions (Bray, 1995; Hellingwerf et al., 1995) and transcriptional networks (Buchler et al., 2003) can be viewed as performing neural network computation.

The challenge of implementing neural networks in chemical reaction networks also has a long history. For example, Hjelmfelt et al (Hjelmfelt et al., 1991) propose a binary-valued chemical neuron, whose switch-like behavior relies on competition between excitation and inhibition.

Recently, Moorman et al (Moorman et al., 2019) proposed an implementation of ReLU units based on a fast bimolecular sequestration reaction which competes with unimolecular production and degradation reactions. In contrast to the prior work, our implementation relies solely on the stoichiometric exchange of reactants for products, and is thus completely independent of the reaction rates. Our CRN is also more compact, using only a single bimolecular reaction per neuron, with two species per every connection (without any additional species for the neuron itself). Finally, in contrast to the prior schemes, our CRN converges to a static rather than a dynamic equilibrium, which means that all reactions cease firing. This implies that our implementation does not waste energy to maintain state.

We use neural networks as a way to program chemistry. The programming is done offline in the sense that neural networks are trained in silico. However, there is a body of work on creating chemical systems that are capable of learning in chemistry (Blount et al., 2017; Chiang et al., 2015). Although these constructions are much more complex than ours, and arguably difficult to realize, they demonstrate the proof-of-principle that chemical interactions such as those within a single cell are capable of brain-like behavior.

Besides the above-mentioned theoretical work on chemical neural networks, wet-lab demonstration of synthetic chemical neural computation argues that the theory is not vapid and that neural networks could be realized in chemistry. A chemical linear classifier reading gene expression levels could perform basic disease diagnostics (Lopez et al., 2018). Larger systems based on rate-dependent strand displacement cascades were used to implement Hopfield associative memory (Qian et al., 2011), and winner-take-all units to classify MNIST digits (Cherry & Qian, 2018).

6. Conclusion

We demonstrate how BinaryConnect (weight ± 1) neural networks could be implemented in chemistry using rate-independent chemical reaction networks. As proof of principle, we demonstrate our scheme with numerical simulations of resulting CRNs classifying the MNIST and IRIS datasets. We further simulate a CRN constructed from a ReLU network trained to classify a virus gene expression dataset. Since this network relies on chemically available information for input, this example argues for the potential biological and medical utility of programming chemical computation via a translation from neural networks.

Performing the simulations with rate constants on the order of magnitude of those reported for DNA strand displacement (10^3 to 10^4 M/s) (Srinivas et al., 2017) results in roughly the conversion factor of 1 time unit = 100 to 1000 seconds of real time (assuming 10^{-6} M concentrations). Although molecular systems may be orders of magnitude slower than electronics, the time scales are compatible with the intended applications such as interfacing with biological regulatory networks or disease marker detection. As a point of comparison, other implementations of chemical neural networks also take hours to complete (Cherry & Qian, 2018).

The convergence time of the CRNs described here scales in a complicated way with many variables, from the depth of the implemented neural network to the choice of representing input values in dual-rail representation (an input $x = 3$ can be represented for example as $x^+ = 3$ and $x^- = 0$, or as $x^+ = 4$ and $x^- = 1$). Thus an important area of future research is the formal proof of convergence properties of the system. Empirically, we observed that depending on the network architecture, the 90%-completion time may increase sublinearly, linearly, or superlinearly (in some contrived examples) with the number of layers.

Note that the concentrations of the positive (S^+) and negative (S^-) dual-rail species can become very large, despite the value they represent remaining small. This can occur in special cases of the networks we construct. This is undesired generally; in synthetic implementations, reactions can use additional "fuel" species in addition to the species designed in the formal CRN, so higher concentrations require more of such species. However, there is an easy fix to this problem: inclusion of the reaction $S^+ + S^- \longrightarrow W$, where W is some inert waste species. This cancellation reaction leaves the signal value unchanged and reduces the concentrations of the two species.

Although in principle arbitrary CRNs can be implemented using DNA strand displacement reactions, current laboratory demonstrations have been limited to small systems (Srinivas et al., 2017), and many challenges remain in constructing large CRNs in the laboratory. Rate independent CRNs offer an attractive implementation target due to their absolute robustness to reaction rates.

Our construction is surprisingly compact in the sense that we use exactly one reaction per ReLU node. This compactness argues that neural networks may be a fitting paradigm for programming chemical computation. This invites the exchange of knowledge between the molecular programming and deep learning communities.

Acknowledgements

References

Anderson, E. The species problem in iris. *Annals of the Missouri Botanical Garden*, 1936.

Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M. J., and Peralta, R. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 18(4):235–253, 2006.

Blount, D., Banda, P., Teuscher, C., and Stefanovic, D. Feedforward chemical neural network: An in silico chemical system that learns xor. *Artificial life*, 23(3):295–317, 2017.

Bray, D. Protein molecules as computational elements in living cells. *Nature*, 376(6538):307–312, 1995.

Buchler, N. E., Gerland, U., and Hwa, T. On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences*, 100(9):5136–5141, 2003.

Chalk, C., Kornerup, N., Reeves, W., and Soloveichik, D. Composable rate-independent computation in continuous chemical reaction networks. In Ceska, M. and Safránek, D. (eds.), *Computational Methods in Systems Biology - 16th International Conference, CMSB 2018, Brno, Czech Republic, September 12-14, 2018, Proceedings*, volume 11095 of *Lecture Notes in Computer Science*, pp. 256–273. Springer, 2018. doi: 10.1007/978-3-319-99429-1_15. URL https://doi.org/10.1007/978-3-319-99429-1_15.

Chen, H.-L., Doty, D., and Soloveichik, D. Deterministic function computation with chemical reaction networks. *Natural Computing*, 13(4):517–534, 2014a.

Chen, H.-L., Doty, D., and Soloveichik, D. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pp. 313–326, 2014b.

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., and Temam, O. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014c.

Chen, Y.-J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., and Seelig, G. Programmable chemical controllers made from DNA. *Nature nanotechnology*, 8(10):755, 2013.

Cherry, K. M. and Qian, L. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370–376, 2018.

Chiang, H.-J. K., Jiang, J.-H. R., and Fages, F. Reconfigurable neuromorphic computation in biochemical systems. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 937–940. IEEE, 2015.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

CRNSimulator. Mathematica package for working with networks of coupled chemical reactions. http://users.ece.utexas.edu/~soloveichik/crnsimulator.html.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Fages, F., Le Guludec, G., Bournez, O., and Pouly, A. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In *International conference on computational methods in systems biology*, pp. 108–127. Springer, 2017.

Fisher, R. A. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 1936.

GEO2R. Identifying Differentially Expressed Genes. https://www.ncbi.nlm.nih.gov/geo/geo2r/.

GSE73072. Host gene expression signatures of H1N1, H3N2, HRV, RSV virus infection in adults. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE73072.

Hellingwerf, K. J., Postma, P. W., Tommassen, J., and Westerhoff, H. V. Signal transduction in bacteria: phospho-neural network(s) in Escherichia coli? *FEMS microbiology reviews*, 16(4):309–321, 1995.

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Karp, R. M. and Miller, R. E. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

Kim, S. K., McAfee, L. C., McMahon, P. L., and Olukotun, K. A highly scalable restricted boltzmann machine fpga implementation. In *2009 International Conference on Field Programmable Logic and Applications*, pp. 367–372. IEEE, 2009.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Li, F., Zhang, B., and Liu, B. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

Lopez, R., Wang, R., and Seelig, G. A molecular multi-gene classifier for disease diagnostics. *Nature chemistry*, 10(7):746–754, 2018.

Moorman, A., Samaniego, C. C., Maley, C., and Weiss, R. A dynamical biomolecular neural network. In *58th IEEE Conference on Decision and Control*. IEEE, 2019.

Petri, C. A. Communication with automata. 1966.

Qian, L., Winfree, E., and Bruck, J. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

Senum, P. and Riedel, M. Rate-independent constructs for chemical computation. In *Biocomputing 2011*, pp. 326–337. World Scientific, 2011.

Severson, E. E., Haley, D., and Doty, D. Composable computation in discrete chemical reaction networks. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC 19, pp. 1423, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362177. doi: 10.1145/3293611.3331615. URL https://doi.org/10.1145/3293611.3331615.

Simons, T. and Lee, D.-J. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

Soloveichik, D., Cook, M., Winfree, E., and Bruck, J. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

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

Srinivas, N., Parkin, J., Seelig, G., Winfree, E., and Soloveichik, D. Enzyme-free nucleic acid dynamical systems. *Science*, 358(6369):eaal2052, 2017.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

Vasic, M., Soloveichik, D., and Khurshid, S. CRN++: molecular programming language. In *International Conference on DNA Computing and Molecular Programming*, pp. 1–18, 2018.

Vasic, M., Soloveichik, D., and Khurshid, S. CRNs exposed: Systematic exploration of chemical reaction networks. *arXiv preprint arXiv:1912.06197*, 2019.