Feedforward Chemical Neural Network: A Compartmentalized Chemical System that Learns XOR

Drew Blount; Peter Banda; Christof Teuscher, *Senior Member, IEEE*; and Darko Stefanovic, *Senior Member, IEEE*

Abstract—The prospects of molar-scale parallelization and biologically-embedded computers make the chemical medium appealing for computation, but reprogrammable chemical computers have not yet been realized. We present the Feedforward Chemical Neural Network (FCNN), the first model of a chemical neural network capable of learning. It is an adaptive and programmable chemical computer.

Our system is built in a simulated chemical environment defined by realistic rate law equations. The network learns by a method akin to backpropagation, ~~and~~ end-user programming is as simple as providing inputs, ~~and ne~~gative reinforcement when the output is incorrect. By this method, one FCNN can be reprogrammed indefinitely.

A single chemical perceptron forms the building block of our network, which is constructed by housing chemically identical neurons in individual cellular compartments. Signals propagate between neurons via permeation through the cell walls. The FCNN correctly learns the binary, two-input exclusive OR; to the best of our knowledge, it is the first simulated chemical system to learn any linearly inseparable function.

Index Terms—Artificial chemistry, learning, chemical computing, feedforward, compartmentalization, backpropagation.

I. Introduction

Just as neural networks were inspired by models of the brain, and a desire to emulate and understand its computational power, chemical computation looks to the cell and other biological systems, hoping to comprehend and harness the information-processing dynamics which drive life [1]–[3]. Within every living cell, there is a powerful chemical computer, capable of dynamic resource allocation, repair, reproduction, and sometimes even motility. We wish to understand chemical computation, in particular, we wish to know if synthetic chemical networks capable of learning and adaptation can be built, and how. In designing such networks, we look to neural networks as inspiration. Chemicals make up neurons, the biological cells which drive brain activity, but can neuronal behavior be emulated in a purely chemical, abiological medium?

Decentralized computation comes naturally to chemical systems. A chemical bath contains an immense number of individual molecules—on the molar scale of 10^{23}—each following the same rule-set and interacting only with its neighbors. These local interactions give rise to system-wide, macroscopic phenomena, such as changes in color, temperature, or state of matter. We wish to construct a system where moles of molecules react in parallel to perform massively decentralized computations.

Further, a programmable 'wet' computer could have myriad medical applications, particularly in smart drug delivery [4]: we imagine a tiny chemical computer injected into a patient, perhaps in its own cellular membrane, that dynamically reasons about the chemical composition of the patient's blood and releases drugs as appropriate. This is a particularly potent option considering that such chemical computers would likely be implemented using biocompatible manufactured DNA strands [5].

There have been many projects on developing chemical computers in simulated environments over the past two decades [1], [6]–[9]. Several models of chemical neural networks have been explored [10]–[13], but so far these networks have been static in their behavior, incapable of learning.

In our previous work we proposed several binary and analog chemical perceptrons [14]–[16]. These were the first simulated chemical systems capable of fully autonomous learning. We used Chemical Reaction Networks (CRNs), which are unstructured macroscopic chemistries, where interactions between symbolic species follow realistic reaction-kinetic laws. In these CRNs, single perceptrons were able to learn two-input linearly-separable functions, and also tackled time-series learning and prediction by introducing chemical delay lines, which store past inputs (concentrations) and feed them to underlying perceptrons [17], [18].

Having developed a family of individual chemical perceptrons, we wish to design a method for connecting these in a more computationally-powerful network. The network should be modular, such that networks with different topologies are constructed from different combinations of the same parts. As neural networks have been shown to be powerful machine learners, we hope that a network of single chemical perceptrons could be a step towards the first general-use reprogrammable chemical computer.

We achieve this goal with the *Feedforward Chemical Neural Network (FCNN)*, a network of cellular compartments, each containing a chemical neuron as a module. Communication between nodes in the network is achieved by permeation through the walls of these compartments, facilitating the network's feedforward and backpropagation mechanisms.

Like standard single-layer perceptrons, each of our individual chemical perceptrons can learn the linearly separable binary two-input logic functions, such as AND and OR, but they are incapable of learning the linearly inseparable functions XOR and XNOR [19]. Here, we demonstrate that the FCNN learns each of these functions. To our knowledge, it is the first chemical system able to learn a linearly inseparable function.

The FCNN presented here has the simplest feedforward neural network topology: one hidden layer with two hidden neurons, the same as the first classical neural net to learn XOR via backpropagation [20]. For more complex learning problems, we show how the FCNN's modular design can be applied to topologies with more, or larger, hidden layers. In any case, using an FCNN is as simple as injecting a few species into its outermost chemical container, and measuring the concentration of an output species. By modulating the concentrations of injected species, different inputs can be provided, the network can be trained to new tasks, and its learning rate can be annealed.

Built on a realistic model of chemical dynamics, the FCNN ~~Feedforward Chemical Neural Network~~ is a step towards reusable, reprogrammable chemical computers. The FCNN, or something very similar, will likely form the basis of the first full-featured neural networks to be built in a purely chemical medium. Bringing computation into the chemical domain will not only change medicine, but computer science and biology as well. By implementing human-programmed computation in biochemical settings, we will also be a step closer to understanding the information-processing that has always occurred at the cellular level of life. The FCNN shows that a relatively simple chemical reaction network can perform, and learn, complex computation.

II. Chemical Formalism

A. The Chemical Reaction Network

Our simulated chemical system consists of chemical species and reactions between them, along with rate coefficients governing the speed of those reactions. In this model, also known as a *Chemical Reaction Network* (CRN), there is no notion of space, as each species is considered only in terms of its concentration in a well-stirred chemical bath, i.e., one where each species' concentration is uniform throughout the mixture. There is no concept of molecular structure—species' behavior is entirely described by the reaction networks. Reaction dynamics are deterministically driven by the concentrations of the chemical species and the reaction coefficients.

Our CRN uses both catalytic and non-catalytic reactions. Non-catalytic reactions follow the mass action law [21], [22], which states that the rate of a chemical reaction is the product of the concentrations of the reactants and a (positive) reaction rate k. For the reaction $S_1 + S_2 \rightarrow P$, where substrates S_1 and S_2 combine to form product P, the reaction rate $d[P]/dt$ is

$$\frac{d[P]}{dt} = -\frac{d[S_1]}{dt} = -\frac{d[S_2]}{dt} = k[S_1][S_2]. \qquad (1)$$

Square brackets around a species symbol denote the concentration of that species in the system.

In catalytic reactions, a chemical species (the catalyst) drives the reaction without being consumed by it. If substrate S turns into product P in the presence of catalyst E (denoted $S \xrightarrow{E} P$), so-called Michaelis-Menten kinetics [23], [24] prescribe the reaction rate,

$$\frac{d[P]}{dt} = \frac{k_{cat}[E][S]}{K_m + [S]}, \qquad (2)$$

where k_{cat}, $K_m \in \mathbb{R}^+$ are rate constants.

We allow 'annihilation reactions,' which remove chemicals from the system. These reactions work as any other, but produce the null product λ, which does not signify disappearance of matter but simply an inert species that does not further interact with other species in the system.

These different types of chemical reactions then give rise to a system of Ordinary Differential Equations (ODEs) over the concentrations of species within our system. This system of ODEs mathematically defines our chemical reaction network.

B. Cellular Compartments

Since Varela and Maturana [25], cellular compartmentalization has been considered a central characteristic of living systems. The analogy of the cell as a self-contained, self-sustaining regulatory machine is a familiar one, and the most basic requirement for such a machine is compartmentalization—separation from the outside world. Several important philosophical explanations of biological organization consider this concept of closure to be essential [26]. Hence, much effort has gone into determining the salient characteristics of a cell, and simulating cells in the field of artificial life [27]–[30]. For these reasons, cells have been a common component of several significant accomplishments in chemical computing [1], [31], [32].

In our chemical system, we use cellular walls as containers for the individual chemical perceptrons, which compose the FCNN. Interaction between neurons is facilitated by rules of permeation across the membranes of these cells. We refer to these permeation rules as *channels*.

Channels can be either reactive or inert. In the first case, a species is allowed to enter the cellular wall, but a different species emerges on the other side. Chemically, we imagine a molecule which reacts with the cell wall as it passes through it. A species passing through an inert channel is not changed; it simply travels from one side of the membrane to the other. A given cell wall can have any number of channels of either type.

Unlike the reactions in our CRN, whose kinetics are driven by simple but chemically accurate Equations 1 and 2, there is no obvious and simple choice for a model of permeation kinetics. In nature, cell walls, membranes, biofilms, and similar structures exist in a number of different forms. There are numerous models describing the permeability of these structures in a variety of different contexts and levels of detail [30], [33]–[35].

In modelling membrane permeation, it is common to consider the pressure inside the cell, or, in a similar vein, the numbers of particular molecules within it [30], [35]. In the latter case, a chemical species S permeates into a cell more slowly if there is already a high concentration of S in the cell. Furthermore, if a cell's total volume is constrained, species can permeate into it only up to a certain point, when the cell becomes 'full.' More detailed models also consider solute permeability, viscosity, hydraulic conductivity [33], pH, and temperature.

We make several simplifying assumptions about our cells. First, the cell's internal pressure is constant, meaning either that the total volume of permeation is much smaller than that of the cell, or that the cell's volume can dynamically expand and contract to maintain pressure. Second, permeation rules are inherently one-way; if a species passes from side A of a cell wall to side B, it does so aware only of the state on side A. This means that permeation is not osmotic, or equilibrium-seeking. Third, since we consider chemical species only in terms of their concentrations, other physical parameters such as temperature and viscosity are beyond the scope of our model. Having thus simplified the chemical picture, channel permeation rates follow mass-action kinetics: the rate of permeation is exactly proportional to the concentration of the source species—in the source container—and a permeability constant k.

We introduce a notation for channels. Consider two example channels through the wall C,

$$\begin{aligned} &\mathrm{C}:(S_2 \leftarrow S_1) \text{ and} \\ &\mathrm{C}:(S_1' \rightarrow S_2'). \end{aligned} \tag{3}$$

The first and second species within the parenthesis are always inside and outside of C, respectively, and the arrow denotes the direction of permeation. Here $\mathrm{C}:(S_2 \leftarrow S_1)$ is a reactive channel in which S_1 passes into C, turning into S_2 in the process.

Cells and their membranes are the central object of study in the field of P Systems, which considers heavily abstracted chemical systems where different reactions occur in different cells, with communication between cells via membrane permeation [36], [37]. The work presented here is superficially similar to P Systems, so it is important to understand key differences between the two models. Our approach to chemistry is less abstracted than that taken in P Systems: the chemical objects of P Systems are strings that 'react' through rewriting rules akin to Chomsky's context-free grammars [36]. P Systems successfully demonstrated the computational power of formal grammars of a specific type, but significant omissions, such as any kinetic model, separate them from chemical computing in practice.

C. Theoretical Strength

Our artificial chemistry [9] constitutes a logic built upon the ordinary differential equations 1 and 2. Our task is to construct autonomously learning neural networks from systems of these ODEs, so our chemical perceptrons are mathematically distinct from perceptrons as they are commonly defined formally [38].

Each phase of the operation of a multilayered perceptron, such as 'calculate linear sum,' and 'update weights,' is only qualitatively emulated by our ODEs—we do not aim to reproduce the math of neural networks in a one-to-one fashion in chemistry. Moreover, chemical reactions representing each operation run continuously and in parallel, rather than discretely and in sequence. Because our network is a large, nonlinear system of ODEs, there are generally no analytic solutions for, say, what concentration of Y will be present in our system a set time after it is started with given initial conditions. We therefore use numerical ODE solvers as the backbone of our simulations~~, as discussed~~ in Section V.

Previous results on the theoretical power of perceptrons and neural networks (e.g., Hornik's [illegible] that feedforward perceptrons are universal app[illegible] with the mathematical definitions [illegible] upon assumptions (as basic a[illegible] $\sum w_i x_i$) [illegible] our chemical perceptrons. [illegible]ermore[illegible]ssed in [illegible]ion II-B, our networks are restricted [illegible]ke topo[illegible] a restriction which is generally not ma[illegible] cal perceptrons.

aside + :)

Nonetheless, Rumelhart's classic exposition of backpropagation [20] treats just our special case: solving XOR with a one-hidden-layer, two-hidden-unit feedforward perceptron. In Section V-A we will use this early result from classical multilayer perceptrons as a benchmark against our FCNN.

III. Chemical Neurons

The chemical neuron, which forms the basis of the FCNN, is the *Analog Asymmetric Signal Perceptron* [16] (AASP). we introduce this model and its methods of input-weight integration and autonomous learning in Section III-A. Two new variants of the AASP are used, one for hidden neurons and one for the output neuron, discussed in Section III-B2. Having discussed the neurons, we move on to the network: we specify the compartments containing each single perceptron and the channels between them in Section IV. It is through these permeation channels that the signal feeds forward through the network, and the error propagates backward. As a reference throughout this section and the rest of the paper, see Table I, which lists all chemical species in the FCNN. Tables listing all reactions, reaction rate constants, and permeation channels are provided in the supplementary material.

TABLE I
Chemical Species in the FCNN
*: OCN only; #: HCN only

Function	Species
Inputs	X_1, X_2
Output	Y
Weights	W_0, W_1, W_2
Input (clock) signal	S_{in}
Learning signal	S_L
Production records	$X_1Y, X_2Y, S_{in}Y$
Weight adjusters	$W^{\oplus}, W^{\ominus}$
Indiv. weight decreasers	$W_0^{\ominus}, W_1^{\ominus}, W_2^{\ominus}$
Inert input transmit*	X_1', X_2', S_{in}'
Binary threshold*	T
Penalty*	P
Error signal*	$E^{\oplus}, E^{\ominus}$
Backprop signals*	$P_1^{\oplus}, P_1^{\ominus}, P_2^{\oplus}, P_2^{\ominus}$
Feedforward signal#	S_F
Feedforward output#	F

– abbreviation / italics policy ?

A. Our Chemical Neuron: The AASP

The *Analog Asymmetric Signal Perceptron* (AASP) forms the basis of our network. Our previous work [16] introduced the AASP and described its mechanism and motivation in depth. Because both our hidden and output neurons (described in Sections III-B1 and III-B2) are modified versions of the AASP ~~Analog Asymmetric Signal Perceptron~~, we describe the relevant features of the AASP to the FCNN ~~Feedforward Chemical Neural Network~~ in this section. Here we present the two-input AASP, though the model can be easily extended by treating any further inputs analogously to the first two.

1) Input: Each component of the input vector is represented by a species X_i. Though the AASP accepts continuous input values, it also operates very well as a binary-input perceptron. In this case, we inject only those X_i whose value is 1, at a preset input concentration, and do not inject the X_i whose value is 0.

Under this input encoding, the zero input ($X_i = 0$ for all i) corresponds to no chemical injection, which poses problems for a system that should react to such an input—the zero input is indistinguishable from no input at all. We therefore include a special clock signal S_{in} with every input, alerting the perceptron to the input event. Though S_{in} is necessary only to recognize the zero input, it is included with all inputs for the sake of consistency. We will see later that S_{in} is also useful in the weight integration.

2) Input-Weight Integration: Here we encounter a common problem in mapping mathematical structures to chemical systems: the representation of negative numbers. As chemical species cannot exist in negative concentrations, an arbitrary real number, such as a perceptron's input weight, cannot be represented by a straightforward mapping to a concentration of a single species.

In our earliest models of chemical perceptrons, we solved this problem by associating each weight with two species, one to encode positive weight values, and one negative [14]. To reduce the size and complexity of the chemical system, we have since used a scheme that requires only one species per weight. For this we must sacrifice the straightforward linear relationship between the weight of a formal perceptron and concentrations of weight species in a chemical system—hence the name 'Asymmetric' Signal Perceptron. Our ASP ~~Asymmetric Signal Perceptron~~ is not only simpler than its predecessors, but learns more accurately [15].

The ASP and its descendants (including the AASP and the chemical neurons used in this paper) solve the negative number problem by emulating the functional role of weights in perceptrons without formally replicating the exact mathematical relations.

Like a formal perceptron's weights, our chemical perceptrons' weights determine the persistent state of the system, and when adjusted, modulate the mapping from input to output. With reactions between the weight species W_i, input species X_i, and output species Y, we wish to qualitatively reproduce the simple linear sum,

$$y = w_0 + w_1x_1 + w_2x_2, \tag{4}$$

in such a way that it reproduces the functionality of negative weights.

What we require of each W_i is simply that, in the presence of X_i, the output Y is either increased or decreased depending on the concentration of W_i. Thi[illegible] achieved by a race between two reactions that consume X_i. [illegible] the first, W_i [illegible]talyzes X_i's transformation into Y. In the second, X_i [illegible]nd Y annihilate.

$$\begin{aligned} X_i &\xrightarrow{W_i} Y [illegible] \\ X_i + Y &\rightarrow [illegible] \end{aligned} \tag{5}$$

Note that the first reaction, whi[illegible]duces Y a[illegible]nc-tion of X_i and W_i, simultaneously pro[illegible]ces a record[illegible]ing species X_iY. This species is later used in the weight-update stage of learning, as will be described in Section III-A3. Since the clock species S_{in} is already present in every injection, it acts as the constant-one coefficient of the bias W_0. In terms of equation 5, $S_{in} = X_0$.

Recalling the reaction rate laws in Eqs. 1 and 2, we see that $[Y]$'s rate of change $d[Y]/dt$ during the input-weight integration is the sum of two terms for each input:

$$\frac{d[Y]}{dt} = \sum_i \left(\frac{k_{c,i}[X_i][W_i]}{K_{m,i} + [X_i]} - k_{a,i}[X_i][Y] \right), \tag{6}$$

where each k is a reaction rate constant. The terms inside the sum are the rates of the reactions in Eq. 5; their signs are opposite because one reaction produces Y and the other consumes it. Because the first reaction's rate is proportional to $[W_i]$, a large $[W_i]$ will result in the first reaction producing Y faster than the second reaction consumes it. In this case, the net effect of the two reactions is to increase $[Y]$. When $[W_i]$ is small, the second reaction dominates and $[Y]$ decreases. Thus, the concentration of the weight species W_i determines if the presence of X_i serves to increase or decrease the final output.

Note that upon input injection, each W_i, X_i pair is simultaneously producing and annihilating the same Y. A consequence of this is another disanalogy between chemical and formal perceptrons: weight strengths are relative, as each copy of the second reaction above will proceed at a rate proportional to $[Y]$, which is constantly changing in a manner dependent on every input weight. This sharing of Y by the various inputs, as well as the two reactions above, is diagrammed in Figure 1.

Note that there is no analytic solution to $[Y]$ in the above equation, hence our use of numeric ODE solvers, discussed in Section V. Though we cannot determine the nonlinear dynamics of $[Y]$ without running an ODE solver, the nature of the chemical reaction network enforces a lower bound of zero and an upper bound equal to the sum of the input concentrations, $\sum_i [X_i]$. The upper bound is asymptotically reached as weights go to infinity.

3) Learning: Chemical implementations of the input-weight integration are not new [10], [13], but our previous work was the first to implement autonomous learning [14]. Though each of our neurons has a slightly modified learning process from the AASP, explaining the AASP's will lay the groundwork for learning in chemical neurons. In an AASP, learning starts with the injection of a 'desired output' species

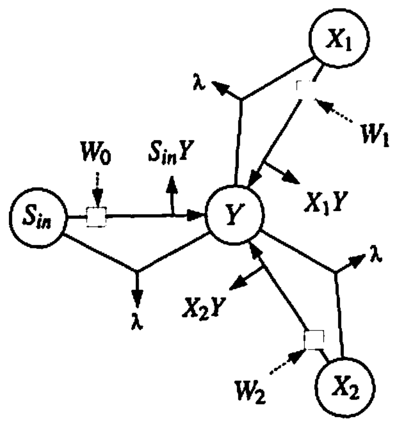


Fig. 1. Reactions in the input-weight integration of a two-input AASP. Dotted arrows denote a catalyst, e.g., signifying $X_i \xrightarrow{W_i} Y + X_iY$. Each weight catalyzes a reaction turning its associated input into Y, while that input and Y simultaneously annihilate each other at a different rate. Reproduced from our paper on the AASP [16].

$\hat{Y}$, and ultimately updates each weight in the appropriate direction by a magnitude relative to that input dimension's impact on the most recent output Y and an analog of error. This scheme is described below, and illustrated in its entirety in Fig. 2. how much? determined how?

A preset time after injecting the input species, the concentration of Y in an AASP is read as its output. Determining the correctness of this output is the first step in the AASP's learning process. This is accomplished by injecting the 'desired output' species $\hat{Y}$ at the concentration we desire of Y. Upon this injection, Y and $\hat{Y}$ quickly annihilate each other in equal proportions via the reaction,

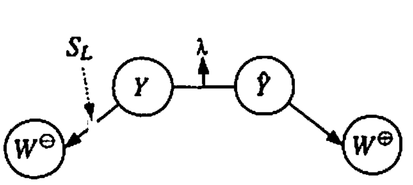
$$Y + \hat{Y} \to \lambda. \tag{7}$$

This reaction consumes all of whichever species has the lower initial concentration, and the remaining species' concentration will be equal to the difference in initial concentrations, i.e., it will be an analog of the error. We then use whichever species remains to create weight-changing species with the appropriate sign, in a slower reaction than the above (to ensure something akin to 'execution order'):

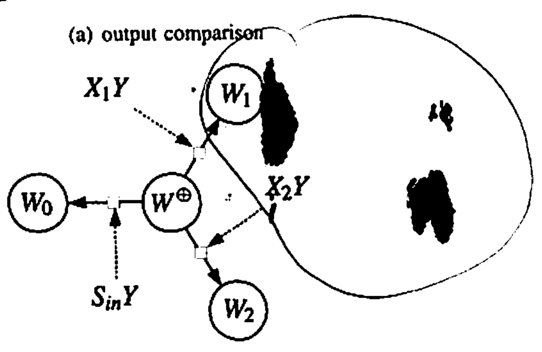
$$\begin{aligned} &\hat{Y} \to W^{\oplus}, \text{ and} \\ &Y \xrightarrow{S_L} W^{\oplus}, \end{aligned} \tag{8}$$

where the learning-signal species S_L, also injected with $\hat{Y}$, ensures that Y is transformed into $W^{\oplus}$ only after $\hat{Y}$'s injection. This process, by which the output is compared with the desired output to produce weight-adjustment species, is illustrated in Figure 2(a). Similar processes are used to calculate error and weight-adjustment signals in both the hidden and output neurons in the FCNN.

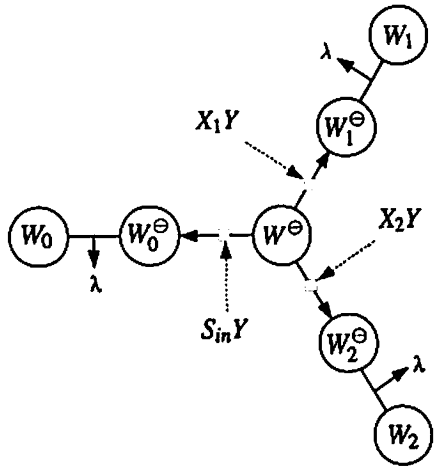
Once weight-adjustment signals are produced, the AASP, HCN, and OCN all behave identically, adjusting their ith ith weight in proportion to both the adjustment signal and the ith input dimension's influence on the most recent production of Y. This qualitatively reproduces the so-called 'delta rule' of classic perceptron learning [38].



(a) output comparison



(b) positive weight adaptation



(c) negative weight adaptation

Fig. 2. The three components of the weight-update mechanism of an AASP. a): Y is compared with $\hat{Y}$ and the appropriately-signed error species is produced. b): The positive error species (the result of $[Y] < [\hat{Y}]$) is transformed directly into each weight species. c): The negative error species is transformed into annihilator-species, which decrease each weight. In both b) and c), the 'record-keeping species' X_iY catalyze the reactions, ensuring that the dimensions that most affected Y will be most affected by the error signal. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

In Section III-A2, we introduced the species X_iY, which is produced as a record of the impact of X_i and W_i on the production of Y. Via catalysis, we use this species to emulate the delta rule: The weight-adjustment species $W^{\oplus}$ and $W^{\ominus}$ adjust the concentration of weight W_i through productive and annihilatory reactions, respectively, each catalyzed by X_iY. Thus, weight-adjustment is achieved by the reactions

$$\begin{aligned} &W^{\oplus} \xrightarrow{X_iY} W_i, \\ &W^{\ominus} \xrightarrow{X_iY} W_i^{\ominus}, \text{ and} \\ &W_i^{\ominus} + W_i \to \lambda. \end{aligned} \tag{9}$$

The main difference between this method and the classical delta rule is that, because X_iY's production is influenced positively by W_i, and X_iY also catalyzes W_i's adjustment, larger W_i

ith ~ i^{th}

will be adjusted relatively more than smaller ones. Thus, larger weights are adjusted more than smaller ones. We reproduce the effect that the *i*th weight is adjusted proportionally to both the difference between desired and actual output, and the most recent *i*th input signal.

Note that it takes an intermediate species and two reactions for $W^{\ominus}$ to annihilate W_i. This is simply because the hypothetical reaction $W^{\ominus} + W_i \xrightarrow{X_iY} \lambda$, with two reactants and a catalyst, would require the collision of three molecules in the same instant. As such three-molecule interactions are unlikely, we do not use them in our designs.

B. Two Breeds of AASP

To accomodate communication between neurons in FCNNs, we had to modify the original AASP design. This resulted in two related breeds: one for hidden neurons, which has a modulated output schedule and accepts backpropagated error signals; and another for the output neuron, modified to initialize the cascading backpropagation reactions. We discuss the means by which these neurons are connected to each other to achieve feeding-forward and backpropagation in Sections IV-B1 and IV-C, but here we first discuss the details that distinguish our output and hidden neurons from each other and the AASP.

1) The Hidden Chemical Neuron: The *Hidden Chemical Neuron* (HCN) is the modular component from which FCNNs of various topologies are constructed. It has two differences from the AASP as presented in Section III-A, one each to facilitate feeding forward and backpropagation.

The AASP produces output Y constantly, as the input dimensions are gradually integrated through the series of reactions in Section III-A2. It is designed to receive input signals instantaneously, however, so in a network context, we cannot simply feed forward a hidden neuron's output as it is produced. We have in practice found that AASPs perform poorly with their input injected gradually over time.

Thus, we introduce a 'feedforward species' S_F, which arrives in an HCN's chemical system when its output Y is meant to feed forward. The details of this will be discussed along with the rest of the network in Section IV-B1, but for now it is enough to know that the following reaction occurs in each HCN:

$$Y \xrightarrow{S_F} F, \tag{10}$$

where F is the species that is later fed forward. Thus the next neuron receives an input signal that is more sudden than the relatively gradual output of the HCN, and this action is induced by S_F.

In terms of learning, an HCN has less work to do than an AASP. An AASP reasons about its output and a desired output to produce the weight-update species $W^{\oplus}$ and $W^{\ominus}$, but hidden neurons receive error signals through backpropagation, not through a direct comparison of the outputs. Thus, the $W^{\oplus}$ and $W^{\ominus}$-producing reactions of an AASP, illustrated in Figure 2(a), are omitted from the HCN.

2) The Output Chemical Neuron: The *Output Chemical Neuron* (OCN) has a different learning mechanism than the AASP. As the current FCNN is designed to learn binary functions, such as XOR, inserting a 'desired output' species to instigate learning, as Figure 2(a) shows, is somewhat ineffective. For example, if the binary threshold is 0.5, the error will be minuscule if the actual output is, say 0.499. This was not a serious problem in the single AASP, but the OCN's error signal must not only update its own weights, but propagate backwards as well. The reactions involving backpropagation will be discussed in Section IV-C; here, we explain the method by which weight-changing signals are produced within the OCN.

The OCN's learning phase begins with the external evaluation of its output. A penalty species P is injected only if the OCN's output is incorrect, i.e., on the wrong side of the binary threshold that is used to interpret chemical concentrations as binary values. Along with P, the AASP's learning signal S_L is injected. Further, a threshold species T is injected in concentration equivalent to the binary threshold. T 'communicates' the binary threshold to the OCN, and behaves somewhat analogously to the 'desired output' species $\hat{Y}$ in the AASP (Section III-A3).

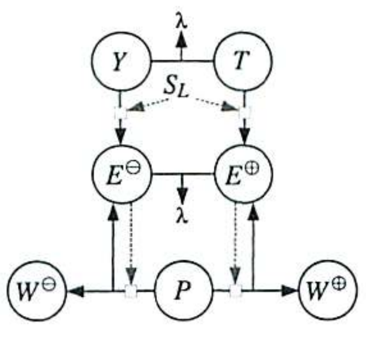


Fig. 3. The weight-update mechanism for a two-input AASP. The process is started by the injection of penalty signal P. The anihilatory comparison of the output Y and the threshold T determines whether the weights will be increased or decreased. Nodes represent species, solid lines are reactions, dashed lines are catalysts, and λ stands for no or inert species.

The goal of the OCN's error-evaluating reactions, diagrammed in Figure 3, is to amplify small continuous-valued errors to emulate distinct binary-valued errors. This is done in two stages. First, in a fast reaction, Y and T annihilate. Whichever is left over encodes whether Y was above or below the threshold, and so if weights should be increased or decreased. So S_L catalyzes relatively slow reactions from Y and T to signed error species $E^{\ominus}$ and $E^{\oplus}$, respectively.

Whichever E species is more prevalent tells whether the weights should be increased or decreased, but their absolute magnitudes might be very small. We then amplify these signals auto-catalytically while consuming the penalty species P:

$$\begin{aligned} P &\xrightarrow{E^{\oplus}} E^{\oplus} + W^{\oplus}, \\ P &\xrightarrow{E^{\ominus}} E^{\ominus} + W^{\ominus}. \end{aligned} \tag{11}$$

Note that $E^{\oplus}$ ($E^{\ominus}$) is both catalyst and product, and $W^{\oplus}$ ($W^{\ominus}$) is also produced. The above equations are illustrated in the bottom half of Figure 3.

The autocatalytic reactions ensure that the total amount of weight-change is not limited by the initial difference between $[Y]$ and $[T]$, which is encoded in the $[E]$s. The total amount of weight-change *is* bounded, however, by the injected concentration of P, as it is the only reactant creating $W^{\oplus}$ or $W^{\ominus}$. Thus, we can achieve annealing by decreasing the concentration of successive injections of P. To summarize the error-evaluating reactions: the initial difference between $[Y]$ and $[T]$ determines the sign of weight-adjustment, ~~but~~ $[P]$ determines the magnitude.

the ? and

IV. Networking

This section discusses the methods by which AASPs are connected to make a FCNN. We first describe the network's topology, both as a neural net and as a series of chemical containers, then its mechanisms of feeding forward and error backpropagation.

A. Constructing the Network Topology

?

With neurons in nested cells and links across cell walls, our networks are topologically trees, with each wall a branch and the outermost cell the root (Figure IV-A). The outermost cell (1 in the figure) corresponds to the output layer in a feedfoward multilayer perceptron, and for a given set of nested cells, the deepest cells (4-7 in the figure) correspond to the input layer. We cannot construct arbitrary feedforward networks, as in our tree-like networks each node can feed forward to only one node in the next layer.

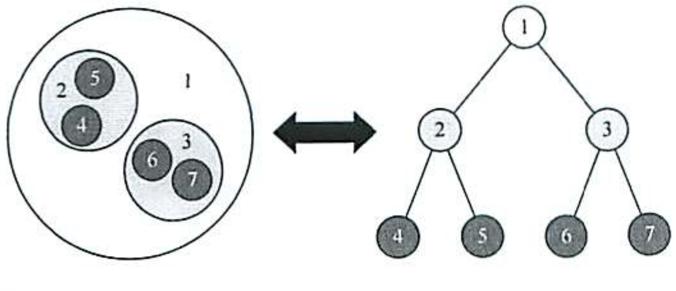


Fig. 4. FCNNs have tree-like topologies.

This nested architecture crucially enables the FCNN's modularity. Each neuron is guaranteed to share a permeable wall with any neuron to which it is connected in the network, so messages which must pass from one neuron to another do not need to be 'addressed'—they passively permeate from parent to child or vice-versa. This allows for scalability. The signal species between different pairs of neurons can be chemically identical, because the signalling occurs in distinct compartments. For this reason, the number of species in the FCNN is sub-linear in the size of the network.

If we wished to make arbitrary feed-forward topologies, it would be possible only if we included distinct signal species for every linkage in the neural network. This could be achieved by placing *all* hidden neurons, regardless of their level in the network, as sibling subcompartments within the larger output neuron. Feedforward and backpropagation signals would travel between hidden neurons via this shared compartment. As long as there are unique signal species for each link in the network, this design allows for arbitrary feedforward network topologies.

not a fan of this, feels like screaming

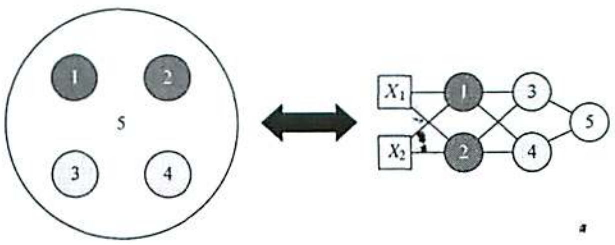


Fig. 5. An alternative scheme places all hidden neurons as sibling subcompartments of the output neuron, illustrated for an ~~an~~ all-to-all network topology.

In the experimental results presented in this paper, we utilized a simple two-hidden-neuron, one-hidden-layer topology to solve XOR. This topology is consistent with either of the above paradigms. As we are more interested in the modularity afforded by the tree-like topology ~~described above,~~ we implement backpropagation and feeding forward in ways generalizable to tree-like designs.

top

B. The Forward Pass

¶ slightly smaller so that fonts match up with other figures

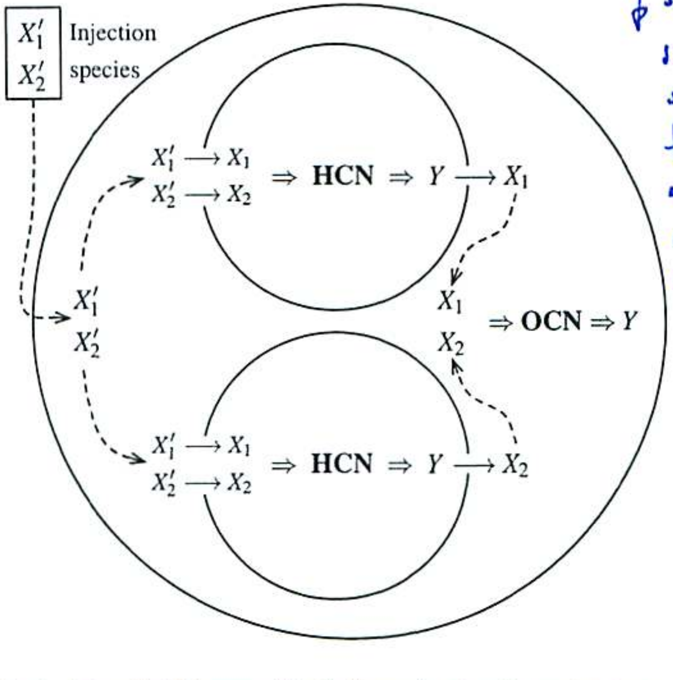

X_1' X_2' Injection species
$X_1' \longrightarrow X_1$
$X_2' \longrightarrow X_2$
$\Rightarrow$ HCN $\Rightarrow Y \longrightarrow X_1$
X_1'
X_2'
X_1
X_2
$\Rightarrow$ OCN $\Rightarrow Y$
$X_1' \longrightarrow X_1$
$X_2' \longrightarrow X_2$
$\Rightarrow$ HCN $\Rightarrow Y \longrightarrow X_2$

Fig. 6. A simplified diagram of the feedforward action of a two-input, one-hidden-layer, two-hidden-neuron FCNN. The inert X_i' species are injected into the outer layer and permeate into the input layer, turning into the reactive X_i input species in the process. Each inner compartment produces Y, which then permeates into the outer compartment as it is transformed into the appropriate X_i. This feedforward process is modulated by unshown species S_F and F, see Section III-B1.

1) Injection: Each of an FCNN's input neurons is contained in a subcompartment of the output neuron. To correspond easily with wet chemistries, all injections into the system are made directly into the outermost compartment, rather than precisely into any of its subcompartments. Yet the input species must be consumed only by reactions in the input-layer nodes, and because the output compartment contains an AASP, any input species X_i are automatically consumed whenever they are present within it. Therefore, an inert species X_i' is injected instead, that permeates into the input compartments. These cells have channels which convert each inert X_i' to the reactive input species X_i. These species are immediately treated by the hidden neurons as input.

$$\mathbf{C} : (X_i \leftarrow X_i') \quad \text{for } \mathbf{C} \text{ an input cell} \tag{12}$$

Similarly, each hidden neuron needs to receive a feedforward species S_F (described in Section III-B1), which must be ultimately injected from outside the system, and every neuron in a given layer should receive this signal simultaneously. In an FCNN with n hidden layers, we require a set of feedforward species $S_F^1, S_F^2, \ldots, S_F^i$, which transform into the basic signal species S_F when permeating into cells of the appropriate layer:

|

ith

$$\mathbf{C} : S_F \leftarrow S_F^i \quad C \text{ in the ith hidden layer.} \tag{13}$$

Once S_F permeates into a given hidden neuron, that neuron's output Y is transformed into the feedforward species F. If this HCN is the ith neuron, in the jth hidden layer, say it is in container ${}^i_j\mathbf{C}$. Then F peremeates outward into $\mathbf{C}$, turning into the corresponding input species X_i in the process:

$$\begin{aligned} &Y \xrightarrow{S_F} F \\ &{}^i\mathbf{C} : (F \rightarrow X_i) \quad {}^iC \text{ is the } \overset{ith}{\text{ith}} \text{ subcell of } \mathbf{C}. \end{aligned} \tag{14}$$

Simultaneously, the OCN receives its signal species S_{in} by recycling the HCN's S_F:

$${}^i\mathbf{C} : (S_F \rightarrow S_{in}) \quad {}^iC \text{ is the } \overset{ith}{\text{ith}} \text{ subcell of } \mathbf{C}. \tag{15}$$

/ |
wp

Thus, S_F plays two roles: it alerts the hidden neurons to feed forward their output, and then by the above permeation alerts the neurons in the next layer ~~forward~~ to begin processing input. Thus the output Y of each hidden chemical neuron feeds forward. This process is illustrated in Figure 7, which shows experimental concentration/time data for the species in the feedforward process, in each neuron in a simple one-hidden-layer FCNN.

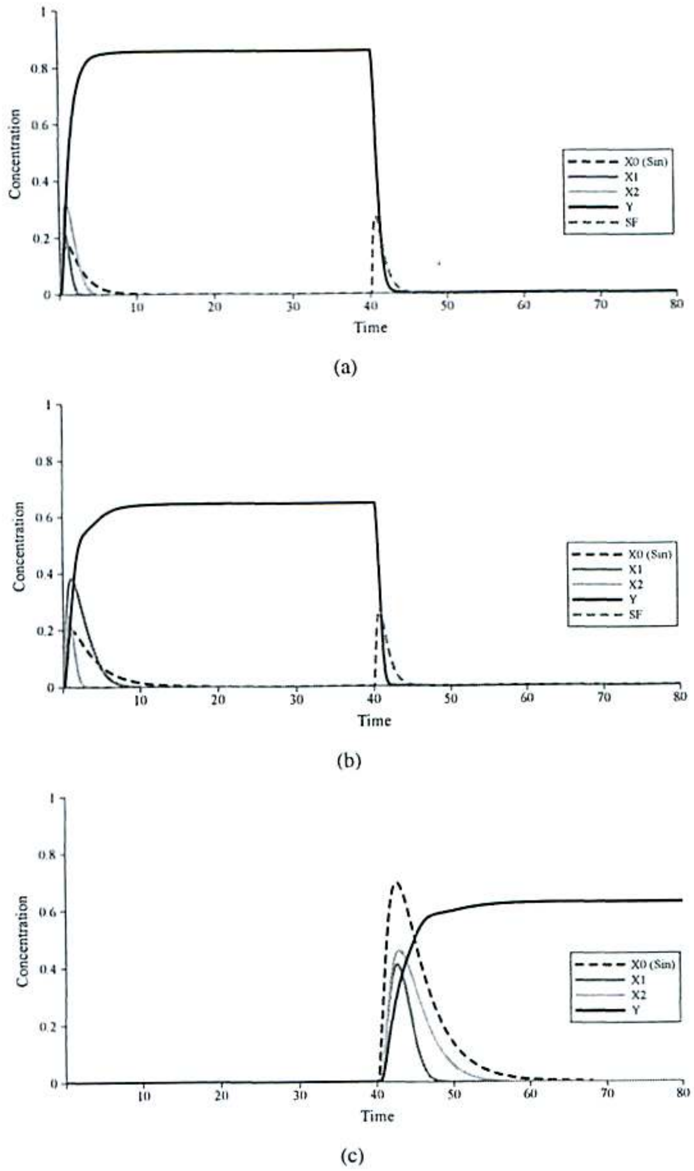
C. Backpropagation

Backpropagation is the algorithm that first learned XOR and popularized the modern feedforward neural network [20]. Again, our chemical system does not exactly reproduce the classic formulae defining this process, and we focus instead on chemically reproducing the important conceptual relationships between parts.

Which one? Chemical or original?

Backpropagation proceeds in three steps:

1) The output perceptron's error $e = \hat{y} - y$ is calculated, where $\hat{y}$ and y are the desired and actual outputs.
2) Moving backwards, each perceptron's error is calculated as the sum of errors which it has affected, weighted by the connecting weights. In our topologically-restricted multilayer perceptrons, each node only feeds forward to one other, so this reduces to $e_i = e_j w_{ji}$, where w_{ji} is the weight from perceptron i to j and e_j is already known.
3) Each weight is updated proportionally to this error, the weight's own magnitude, and the most recent output of the source perceptron: $\Delta w_{ji} \propto e_i w_{ji} y_i$, where y_i is the last output of perceptron i.



(|
lcp

Fig. 7. Concentration/time plot in the example FCNN with two HCNs (a, b) and one OCN (c), illustrating ~~the~~ feeding forward ~~of~~ the HCNs' outputs into the OCN's input. At time zero the inputs X_1', X_2', and S_{in}' are injected to the OCN (not shown in (c)). They then permeate into each HCN, transforming into the input species. Note the initial spikes in concentrations of X_1, X_2, and S_{in} in (a) and (b). After the injection of an S_F signal at time 40, each HCN's output permeates out to the OCN, transforming into the appropriate input species and S_{in}.

The first step above is emulated by the OCN's *internal* reactions, described in Section III-B2. These reactions produce weight-update species that encode the sign and magnitude of the network's overall error.

The OCN generates input-specific error signals to be backslightly smaller to match [illegible] fonts.

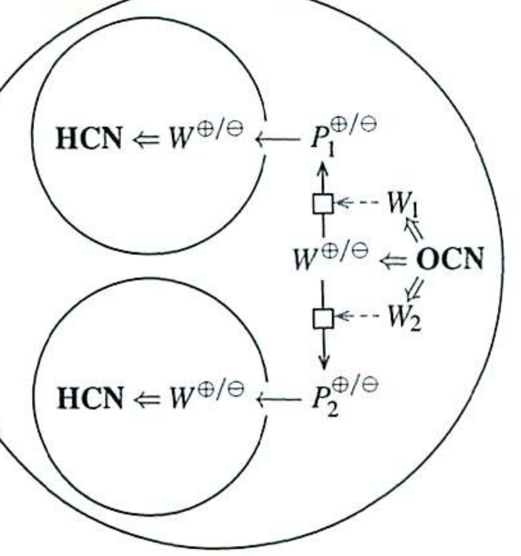

HCN ⇐ $W^{\oplus/\ominus}$ ⟵ $P_1^{\oplus/\ominus}$
W_1
$W^{\oplus/\ominus}$ ⇐ OCN
W_2
HCN ⇐ $W^{\oplus/\ominus}$ ⟵ $P_2^{\oplus/\ominus}$

Fig. 8. A diagram of the backpropagation action of a one-hidden-layer, two-hidden-neuron FCNN. The weight-adjusting species in the outer OCN (right of figure) produce signed, input-specific penalty species P_i. The penalty species then permeate into the hidden neurons' compartments, becoming those neurons' weight-changing species in the process. Six of the functions are labelled; the remaining ten overlap in the top-left of the graph. Note that the FCNN learns equally well any two functions, if they are equivalent after switching the names of X1 and X2.

?

propagated to the previous layer in the network. These reactions are also implemented in any HCN with deeper layers of HCNs inside of it, i.e., all neurons where a signal should be backpropagated. In such neurons, the weight-adjustment species $W^{\oplus}$ and $W^{\ominus}$, in addition to changing weights, produce input-specific backpropagation signals. This production is catalyzed by the weight species W_i,

$$
\begin{aligned}
&W^{\oplus} \xrightarrow{W_i} P_i^{\oplus}, \\
&W^{\ominus} \xrightarrow{W_i} P_i^{\ominus},
\end{aligned} \tag{16}
$$

ith

so the *ith* backpropagation signal $P_i^{\oplus}$ or $P_i^{\ominus}$ is produced in an amount positively correlated with the *ith* weight and the overall adjustment species $W^{\oplus}$ and $W^{\ominus}$.

The signed, dimension-specific penalty species $P_i^{\oplus}$ or $P_i^{\ominus}$, then propagate backwards through the network via permeation channels. Since the purpose of these signals is to adjust weights, they are simply transformed to the species $W^{\oplus}$ and $W^{\ominus}$ as they permeate into the appropriate subcontainers:

$$
\begin{aligned}
&{}^{i}\mathbf{C} : (W^{\oplus} \leftarrow P_i^{\oplus}) \\
&{}^{i}\mathbf{C} : (W^{\ominus} \leftarrow P_i^{\ominus}) \qquad {}^{i}C \textit{ is the ith subcell of } \mathbf{C}.
\end{aligned} \tag{17}
$$

Thus in the one-hidden-layer case, the concentration of the weight-changing species $W^{\oplus}$ and $W^{\ominus}$ in the *ith* inner compartment is related to a) $W^{\oplus}$ and $W^{\ominus}$ in the outer compartment, and b) the weight W_i connecting the two compartments. This relationship is diagrammed in Figure 8.

D. Rate and Permeation Constants

Though we have discussed all of the reactions and permeation channels in the FCNN, we have so far only specified the reactants, products, and catalysts in each reaction, but every reaction has at least one dimensionless rate coefficient which describes its speed (the *k*s in Eqs. 1 and 2). Here we discuss how those rate coefficients have been set in our experiments. Tables listing all reaction rates can be found in the supplementary material.

As each AASP (Section III-A) contains around twenty distinct rate constants, the rate parameter space is prohibitively large and complex to explore via exhaustive search. In our previous work [16] we searched for these constants with a standard genetic algorithm. As both the Hidden and Output Chemical Neurons share most of their reactions with the AASP, those reactions are set to the rates found by the method described in [16].

Unlike with the AASP, we found success in setting the reactions introduced in this paper by hand. Our intuition in doing so was based on the intended function of each reaction, and which reactions should occur relatively faster or slower than others. To illustrate the intuitive setting of rate constants, consider the case when two species annihilate in order to determine which had the larger initial concentration, and then the remaining species is transformed into another to perform a task dependent on the initial comparison. This is the case when Y and T are compared in the OCN's error-calculating mechanism (Fig. 3): whichever species remains after annihilation is meant to turn into an appropriately-signed error species. In cases such as these, the comparison reaction should be faster than the follow-up reactions, dependent on that comparison. Otherwise, the second reaction would execute before the comparison had been meaningfully made. These manually set rate constants, as well as those set by the genetic algorithm, are listed in the supplementary material.

V. Results & Implementation

As our goal is to learn linearly inseparable functions in a chemical neural network, we built the first FCNN in the classic one-hidden-layer, two-hidden-neuron network topology first shown to learn XOR [20]. We will refer to this topology, ~~with~~ rate constants set as described in Section IV-D simply as *the FCNN* in this section.

The FCNN successfully learns each of the sixteen binary two-input logic functions. In a single learning run, the FCNN was trained to a given logic function with 2,000 random inputs and corresponding penalty injections (described in Section IV). We generated inputs randomly from $\{0,1\}^2$ and later injected a threshold species T with concentration 0.6 and a penalty P as appropriate, 2,000 times consecutively. The concentration of the penalty species, which is analogous to a learning rate, was annealed by a factor of 0.0008 at each injection (see the supplementary material for further details). it

The FCNN's accuracy at each of the 2,000 consecutive iterations was averaged over 10,000 training runs, to produce accuracy/time data for each logic function shown in Figure 10. Figure 9 shows an example of the FCNN converging to a solution of XOR.

fonts are bigger than in other figures

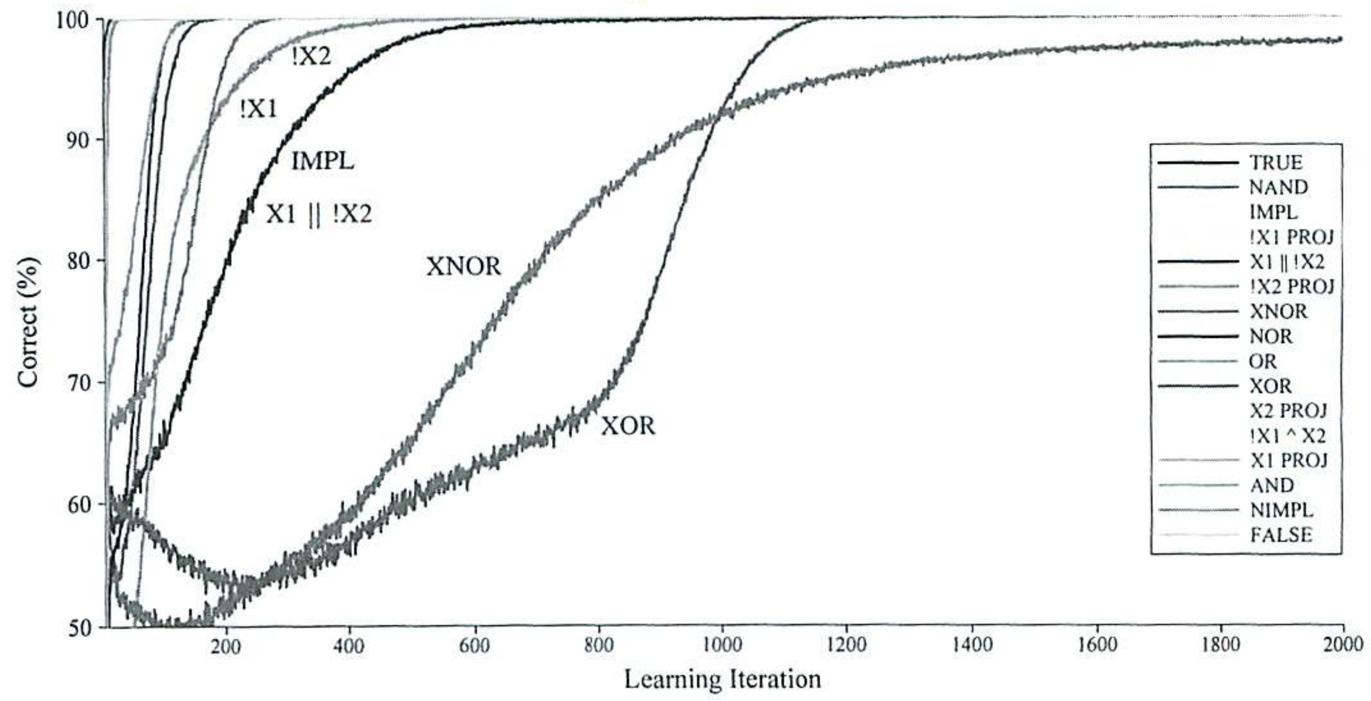


Fig. 10. Average accuracy of the FCNN on each of the 16 binary two-input logic functions. An FCNN with one hidden layer and two hidden neurons layers was run 10,000 times on each of the 16 functions. Each run started with random initial weights and was trained for 2,000 learning iterations. The data points represent the proportion of correct answers the system produced on a given learning iteration.

A. *Performance*

The most important results are the FCNN's performance on XOR and XNOR, which, because of their linear inseparability, cannot be learned by a single perceptron [19]. On the 2,000th iteration, the FCNN's accuracy on XOR and XNOR is 100.00% and 98.05%, respectively. Averaged over all 16 functions, the FCNN is 99.88% accurate by the 2,000th learning iteration.

The difference in performance between XOR and XNOR can be explained by the FCNN's asymmetric treatment of the input values 0 and 1. As can be seen in Fig. 10, the functions $!X1$ and $!X2$ are learned almost identically well by the FCNN. This is because the FCNN's architecture is symmetric on input; each input dimension behaves by the same logic. Its architecture is not symmetric on negation, however, as 1 and 0 values are treated fundamentally differently. Consider the fact that the output species Y is ultimately produced by the two input species X_1 and X_2, as well as the signal species S_{in} (Sections III-A1 and IV-B). For this reason, it is easier for the FCNN to learn to output 0 when it is given the input $(0,0)$ (i.e., when only S_{in} is present at injection), than learn to output 1.

Unlike its building block, the ~~Analog Asymmetric Signal Perceptron~~ ASSP (Section III-A), the FCNN in this context behaves as a binary perceptron. Thus, we compare its performance on the binary logic functions not with the AASP, but with our two best binary single chemical perceptrons: the SASP (Standard ASP) and an automatically thesholded version, the TASP (Thresholded ASP) [15]. The SASP and the TASP represent the previous state-of-the-art in terms of binary chemical perceptrons (though the TASP is more accurate, the SASP is more similar to the FCNN because of its lack of automatic output-thresholding). As shown in Table V-A, the FCNN is significantly more capable than either.

TABLE II
ACCURACY OF FCNN VS. SINGLE CHEM. PERCEPTRONS

Accuracy	FCNN	SASP	TASP
XOR	100.00	50.53	59.00
XNOR	98.05	51.02	57.86
16-function average	99.88	93.4	94.8

Thus, the FCNN is the first chemical system to autonomously learn linearly inseparable functions. As shown by Minsky and Papert [19], single perceptrons cannot learn such functions because, as binary classifiers, they can only divide the input space along a hyperplane. Like a multilayer perceptron, the FCNN does not have this constraint. To illustrate this, we mapped the response surface of an FCNN, which had successfully learned XOR, as shown in Figure 11.

As can be seen in Figure 10, it does take the FCNN relatively longer to converge to learn inseparable functions. The 14 separable functions are almost perfectly learned by the 600th learning iteration, but it takes until around iteration 1,200 to learn XOR. XNOR is not perfectly learned even by step 2,000. We see this as confirmation that linear inseparability is a challenging feature to learn. Nonetheless, the FCNN learns about as quickly as the original multilayer perceptrons that solved XOR: Rumelhart's classic multilayer perceptron took roughly 1,000 learning iterations [20].

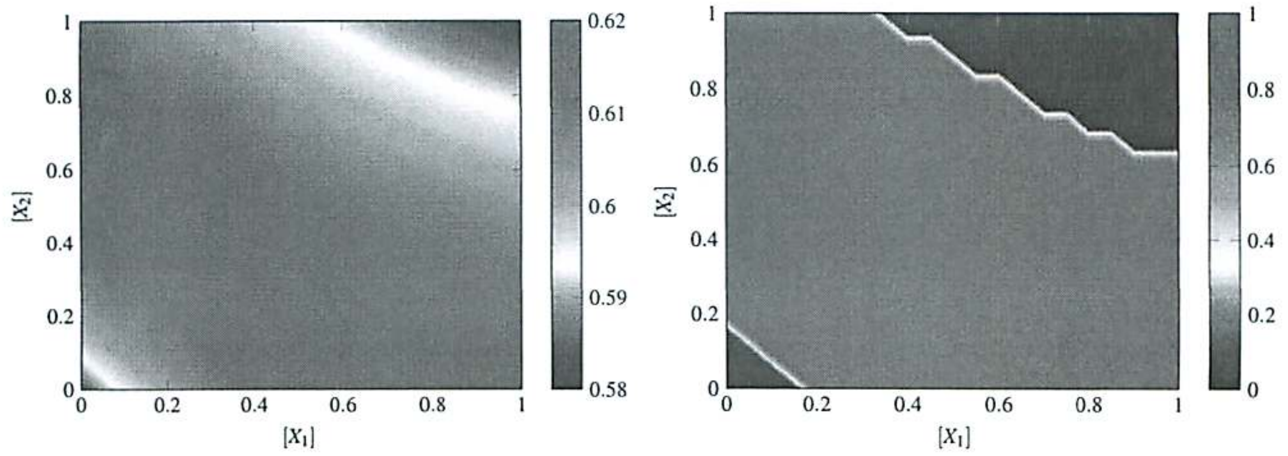
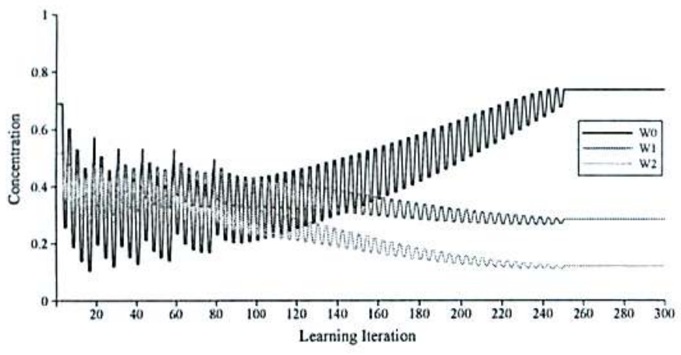


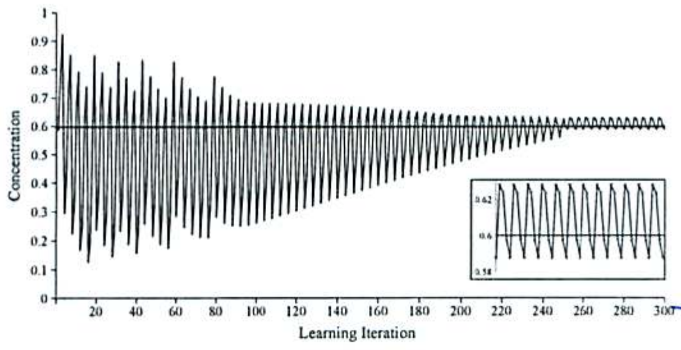
Fig. 11. Response surface of an FCNN which learned XOR. Here input $[X]$ values of 1.0 correspond to TRUE, and 0.0 to FALSE, so the accuracy of the FCNN is defined only by its response at the corners of the plots. The plot on the left shows the FCNN's output value at each $([X_1],[X_2])$, while the plot on the right shows the same data thresholded by 0.6—the output values above the threshold correspond to TRUE (red region), and those below indicate FALSE (blue regions). Jagged lines in the right figure are an artifact of our sampling technique.

✓



(a) weights

✓



(b) output

Fig. 9. An example of weights and output concentration converging in the OCN as an FCNN learns XOR over 300 learning iterations. Note that when the weights reach a fixed point around the 250th iteration, the output $[Y]$ oscillates around 0.6, which in this case is the binary threshold. In this experiment, inputs were cycled in the fixed order $((0,0),(1,0),(0,1),(1,1))$ for the purpose of illustration—once the function is learned, $[Y]$ oscillates as the system produces the correct (thresholded) output stream $0,1,1,0$ (zoomed in the smaller plot).

B. Simulation Details

The FCNN is a large system of ODEs. As such systems are generally unsolvable analytically, we make use of numeric Runge-Kutta-type ODE solvers to observe the FCNN's behavior. We used a fixed-time-step solver with a step size of 0.1, chosen for speed as well as stability. All simulations were run on the COEL web-based chemical modelling and simulation framework, built throughout this and related projects [40].

In the interest of modularity and flexibility of simulations, COEL's ODE solvers do not consider the FCNN as a whole. Rather, the contents of each cellular compartment are simulated semi-independently, with a separate ODE solver in each compartment and the various solvers communicating with each other as needed. This allows us to use different solvers in each compartment, or even to have compartments containing entirely different types of simulations (though this option is unused in our current applications). When using adaptive time step-size ODE solvers, the compartments are synchronized by imposing the "slowest" compartment's time step-size on the other compartments, and so all of the chemical concentrations in the FCNN are updated in unison. Still, this design lends itself naturally to fixed time-step solvers. constant

VI. Conclusion

We have presented a hierarchical, compartmentalized chemical system that is capable of autonomous learning. The ~~Feed-forward Chemical Neural Network~~ FCNN is, to our knowledge, the first chemical system to learn a linearly inseparable function. It does so by a chemical analog of backpropagation. This is demonstrated in the classic example of the XOR binary function, which the FCNN learns perfectly in under 1,500 learning iterations, 100% of the time.

Each chemical neuron in the FCNN is a modified version of the ~~Analog Asymmetric Signal Perceptron~~ ASSP from our previous work. Neurons are distinguished from each other by their compartmentalization in nested cellular walls. This nesting constrains FCNNs to having tree-like topologies, but allows for its design modularity. Inter-neuron communication, facilitating feeding-forward and backpropagation, is mediated by selective permeation of signal species through these cell walls.

Each hidden neuron is chemically identical in terms of the species and reactions that occur within them. This means that the FCNN is easily scalable: once an FCNN with a simple topology has been implemented in wet chemistry, it will be possible to construct much larger networks than the minimal example explored in this paper.

Because they follow realistic reaction rate laws, the chemical reaction networks we have designed (i.e., each individual chemical neuron) are directly translatable to wet chemistries thanks to advances in DNA strand-displacement circuits [5]. Translating our cellular containers into a wet implementation would be more challenging, as the construction of chemical membranes has no straightforward solution, such as DNA strand-displacement. We are hopeful that current work in synthesizing bilayer lipid membranes will develop such a solution.

One technical complication in the FCNN design worth addressing is the manual injection of the feedforward signal-species S_F. Ideally, a chemical neural net could operate completely independently besides receiving input and a desired output signal. We are hopeful of developing mechanisms by which S_F could be generated periodically within the FCNN, perhaps using chemical oscillators such as the Lotka-Volterra predator-prey system [41], [42].

Another direction for further research is optimizing the modules of the FCNN. In our previous work, we have developed a small family of single chemical perceptrons—we have already mentioned the (Standard) Asymmetric Signal Perceptron [15], Thresholded Asymmetric Signal Perceptron [15], and Analog ASP [18]. We have explored network architectures with many of these as building blocks, shown in our supplementary materials. Though we have presented the best-performing architecture we have tested, it is possible that there are yet better components for the FCNN than our current hidden and output neurons.

In our previous work we have developed chemical perceptrons which incorporate delay-lines to learn patterns in time series [17], [18]. Integrating these into an FCNN would add a temporal dimension to the network's pattern-recognition capabilities, which would enable us to tackle standard neural network and reservoir computing benchmark time-series such as NARMA [43], Mackey-Glass [44], [45], and Hénon Map [46]. This is an avenue we plan on exploring.

Another logical extension of this work is to explore the FCNN model on larger networks, with either more hidden layers, more neurons per layer, or both. Larger FCNNs are expected to tackle more complex problems, and it is an interesting and open question how many layers and neurons per layer are necessary for the FCNN to solve certain tasks, as its behavior and performance will likely differ somewhat from classically implemented neural networks.

It bears repeating that the FCNN is not a chemical transcription of a neural network, but an analogy. As discussed with our results in Section V, the FCNN we used here to solve XOR converged about as quickly as Rumelhart, Hinton, and Williams's first XOR-solving network [20]. We wonder if the FCNN generally converges faster or slower than classical neural nets. Our analog of input-weight integration, mathematically written in the language of ODEs, is highly nonlinear—as the building block of a backpropagated network, how does this compare to standard linear integration and sigmoidal activation functions?

The importance of this research is the hope of a reprogrammable chemical computer. Like the formal multilayered perceptron, the FCNN has two modes: simply processing output when given an input, and learning via backpropagation of errors. A wet implementation of the ~~Feedforward Chemical Neural Network~~, once trained to learn a task, could perform that task reliably as long as desired. The chemical computer could then, at any time, be retrained to perform any other task of which it is capable. FCNN

As parallelization becomes ever-more prominent in high-powered computing, it is difficult not to be tempted by the possibility of molecule-scale chemical computers. Such tiny machines could be combined in molar amounts—on the scale of 10^{23}—to perform naturally distributed, parallel computing.

Furthermore, the liquid nature of chemical computers and the possibility of constructing them from bio-compatible DNA strands [5] open profound possibilities for patient-embedded biochemical computers. A fleet of cell-sized machines could monitor chemical concentrations in the bloodstream, modulating the release of some drug accordingly. With time-series integration, biochemical computers could keep a record of changing biological systems and act as diagnostic aids and tools in preventative medicine.

There has already been significant research into medical uses of computational chemical networks. One recent result [47] presented a chemical logic circuit which tests for the presence of micro RNA molecules and the absence of others, effectively identifying a type of cancer cell called HeLa. The authors designed a bespoke chemical logic circuit for this purpose, but this is exactly the type of computation an FCNN excels at learning. A standardized model of a programmable chemical computer would trivialize the design of such medical machines, whose potential has only begun to be explored.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under grant, No. 1028120, 1028238, and 1318833.

REFERENCES

[1] P. Dittrich, "Chemical computing," in *Unconventional Programming Paradigms*. Springer, 2005, pp. 19–32. City?

[2] A. Condon, D. Harel, J. N. Kok, A. Salomaa, and E. Winfree, *Algorithmic Bioprocesses*. Berlin Heidelberg: Springer, 2009.

[3] E. Katz, *Biomolecular Information Processing: From Logic Systems to Smart Sensors and Actuators*. Wiley-VCH Verlag GmbH & Co. KGaA, 2012.

[4] D. A. LaVan, T. McGuire, and R. Langer, "Small-scale systems for in vivo drug delivery." *Nature Biotechnology*, vol. 21, no. 10, pp. 1184–91, Oct. 2003.

[5] D. Soloveichik, G. Seelig, and E. Winfree, "DNA as a universal substrate for chemical kinetics," *Proceedings of the National Academy of Sciences*, vol. 107, no. 12, pp. 5393–5398, 2010.

[6] G. Berry and G. Boudol, "The chemical abstract machine," *Theoretical Computer Science*, vol. 96, no. 1, pp. 217–248, 1992.

[7] K. Tominaga, T. Watanabe, K. Kobayashi, M. Nakamura, K. Kishi, and M. Kazuno, "Modeling molecular computing systems by an artificial chemistry-its expressive power and application," *Artificial Life*, vol. 13, no. 3, pp. 223–247, 2007.

[8] H. Jiang, M. D. Riedel, and K. K. Parhi, "Digital signal processing with molecular reactions," *IEEE Design & Test of Computers*, vol. 29, no. 3, pp. 21–31, Jun. 2012.

Feedforward Chemical Neural Network: A Compartmentalized Chemical System that Learns XOR (Supplementary Material)

Drew Blount, Peter Banda, Christof Teuscher, and Darko Stefanovic

~~October 20 2014~~ Senior Member IEEE

Contained in these materials are the detailed numerical data, defining the FCNN as implemented in this paper. Table 1 lists the cell-wall permeation channels between the neurons in the network, which enable feeding forward and backpropagation. Table 2 lists all chemical reactions and rates in each of our chemical neurons and the AASP. Table 3 lists the ODEs that were integrated to simulate the FCNN, one for each species. Table 4 lists the experimental protocol, i.e., the schedule of external injections into the FCNN that facilitated each learning iteration. Figure 3 shows performance plots of several prototypes of the FCNN, as they attempt to learn the 16 binary functions. These prototypes used neurons derived from different single chemical perceptrons than the AASP, which was our final choice.

1 Model

Table 1: Permeation channels of the two-input, two-layer FCNN, where $^1\mathrm{C}$ and $^2\mathrm{C}$ are the two inner compartments. Groups 1 and 2 enable the inner perceptrons' input-weight integration, groups 3 – 5 the outer perceptron's input-weight integration, and finally, groups 6 and 7 the inner perceptrons' learning (error backpropagation). All permeability constants are set to 1.

Group	Channels
1	$^1\mathrm{C}:(X_1 \leftarrow X_1')$
	$^2\mathrm{C}:(X_1 \leftarrow X_1')$
	$^1\mathrm{C}:(X_2 \leftarrow X_2')$
	$^2\mathrm{C}:(X_2 \leftarrow X_2')$
2	$^1\mathrm{C}:(S_{in} \leftarrow S_{in}')$
	$^2\mathrm{C}:(S_{in} \leftarrow S_{in}')$
3	$^1\mathrm{C}:(S_F \leftarrow S_F)$
	$^2\mathrm{C}:(S_F \leftarrow S_F)$
4	$^1\mathrm{C}:(F \rightarrow X_1)$
	$^2\mathrm{C}:(F \rightarrow X_2)$
5	$^1\mathrm{C}:(S_F \rightarrow S_{in})$
	$^2\mathrm{C}:(S_F \rightarrow S_{in})$
6	$^1\mathrm{C}:(W^{\oplus} \leftarrow P_1^{\oplus})$
	$^2\mathrm{C}:(W^{\oplus} \leftarrow P_2^{\oplus})$
7	$^1\mathrm{C}:(W^{\ominus} \leftarrow P_1^{\ominus})$
	$^2\mathrm{C}:(W^{\ominus} \leftarrow P_2^{\ominus})$
Total 16	

Table 2: The reactions and rate constants of: (a) the original AASP, an analog asymmetric signal perceptron, (reproduced from [15]) (b) the hidden chemical neuron, and (c) the output chemical neuron. The hidden neuron is a modification of the AASP that omits the output–target-output comparison reactions (which are blacked out) and requires an extra reaction (highlighted in dark gray) for feeding forward the output. The output chemical neuron (also a modification of the AASP) uses penalty species P and threshold T for learning binary output, and propagates error backwards using species $P_i^{\oplus}$ and $P_i^{\ominus}$. These species interact in modifications of the AASP's reactions (highlighted in light gray), as well as totally novel ones (dark gray). Note that first 6 reactions in each set implement the input-weight integrations, the rest implement learning. The catalytic reactions have two rates: k_{cat} and K_m. All rate constants are rounded to four decimal places.

(a) AASP

Reaction	Rates
$S_{in} + Y \to \lambda$	0.1800
$S_{in} \xrightarrow{W_0} Y + S_{in}Y$	0.5521, 2.5336
$X_1 + Y \to \lambda$	0.3905
$X_2 + Y \to \lambda$	
$X_1 \xrightarrow{W_1} Y + X_1Y$	0.4358, 0.1227
$X_2 \xrightarrow{W_2} Y + X_2Y$	
$\hat{Y} \to W^{\oplus}$	0.1884
$Y \xrightarrow{S_L} W^{\ominus}$	0.1155, 1.9613
$Y + \hat{Y} \to \lambda$	1.0000
$W^{\ominus} \xrightarrow{S_{in}Y} W_0^{\ominus}$	0.600, 1.6697
$W_0 + W_0^{\ominus} \to \lambda$	0.2642
$W^{\oplus} \xrightarrow{S_{in}Y} W_0$	0.5023, 2.9078
$W^{\ominus} \xrightarrow{X_1Y} W_1^{\ominus}$	0.1889, 1.6788
$W^{\ominus} \xrightarrow{X_2Y} W_2^{\ominus}$	
$W_1 + W_1^{\ominus} \to \lambda$	0.2416
$W_2 + W_2^{\ominus} \to \lambda$	
$W^{\oplus} \xrightarrow{X_1Y} W_1$	0.2744, 5.0000
$W^{\oplus} \xrightarrow{X_2Y} W_2$	
18 reactions, 20 rates	

(b) HCN

Reaction	Rates
$S_{in} + Y \to \lambda$	0.1800
$S_{in} \xrightarrow{W_0} Y + S_{in}Y$	0.5521, 2.5336
$X_1 + Y \to \lambda$	0.3905
$X_2 + Y \to \lambda$	
$X_1 \xrightarrow{W_1} Y + X_1Y$	0.4358, 0.1227
$X_2 \xrightarrow{W_2} Y + X_2Y$	
[redacted]	[redacted]
$W^{\ominus} \xrightarrow{S_{in}Y} W_0^{\ominus}$	0.600, 1.6697
$W_0 + W_0^{\ominus} \to \lambda$	0.2642
$W^{\oplus} \xrightarrow{S_{in}Y} W_0$	0.5023, 2.9078
$W^{\ominus} \xrightarrow{X_1Y} W_1^{\ominus}$	0.1889, 1.6788
$W^{\ominus} \xrightarrow{X_2Y} W_2^{\ominus}$	
$W_1 + W_1^{\ominus} \to \lambda$	0.2416
$W_2 + W_2^{\ominus} \to \lambda$	
$W^{\oplus} \xrightarrow{X_1Y} W_1$	0.2744, 5.0000
$W^{\oplus} \xrightarrow{X_2Y} W_2$	
$Y \xrightarrow{S_F} F$	3.0000, 0.1000
16 reactions, 18 rates	

grey?

(c) OCN

Reaction	Rates
$S_{in} + Y \to \lambda$	0.1800
$S_{in} \xrightarrow{W_0} Y + S_{in}Y$	0.5521, 2.5336
$X_1 + Y \to \lambda$	0.3905
$X_2 + Y \to \lambda$	
$X_1 \xrightarrow{W_1} Y + X_1Y$	0.4358, 0.1227
$X_2 \xrightarrow{W_2} Y + X_2Y$	
$T \xrightarrow{S_L} E^{\oplus}$	0.1155, 1.9613
$Y \xrightarrow{S_L} E^{\ominus}$	0.1155, 1.9613
$Y + T \to \lambda$	5.0000
$W^{\ominus} \xrightarrow{S_{in}Y} W_0^{\ominus}$	0.600, 1.6697
$W_0 + W_0^{\ominus} \to \lambda$	0.2642
$W^{\oplus} \xrightarrow{S_{in}Y} W_0$	0.5023, 2.9078
$W^{\ominus} \xrightarrow{X_1Y} W_1^{\ominus}$	0.1889, 1.6788
$W^{\ominus} \xrightarrow{X_2Y} W_2^{\ominus}$	
$W_1 + W_1^{\ominus} \to \lambda$	0.2416
$W_2 + W_2^{\ominus} \to \lambda$	
$W^{\oplus} \xrightarrow{X_1Y} W_1$	0.2744, 5.0000
$W^{\oplus} \xrightarrow{X_2Y} W_2$	
$P \xrightarrow{E^{\oplus}} E^{\oplus} + W^{\oplus}$	1.0000, 1.0000
$P \xrightarrow{E^{\ominus}} E^{\ominus} + W^{\ominus}$	
$E^{\oplus} + E^{\ominus} \to \lambda$	5.0000
$W^{\oplus} \xrightarrow{W_1} P_1^{\oplus}$	0.3000, 0.5000
$W^{\oplus} \xrightarrow{W_2} P_2^{\oplus}$	
$W^{\ominus} \xrightarrow{W_1} P_1^{\ominus}$	
$W^{\ominus} \xrightarrow{W_2} P_2^{\ominus}$	
25 reactions, 26 rates	

Table 3: The full list of ODEs modelling the hidden (a) and output (b) chemical neurons, whose reaction are shown in Table 2. Note that these ODEs exclude the contributions or consumptions of the channels (Table 1) mediating the communication between the FCNN's compartments. For easier reproducibility we provide the ODEs in Octave/Matlab format, downloadable at `http://coel-sim.org/download`.

(a) HCN

$$\frac{d[S_{in}]}{dt} = -\frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]} - 0.1800[S_{in}][Y]$$

$$\frac{d[X_1]}{dt} = -\frac{0.4358[W_1][X_1]}{0.1227+[X_1]} - 0.3905[X_1][Y]$$

$$\frac{d[X_2]}{dt} = -\frac{0.4358[W_2][X_2]}{0.1227+[X_2]} - 0.3905[X_2][Y]$$

$$\begin{aligned}\frac{d[Y]}{dt} = {} & \frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]} + \frac{0.4358[W_1][X_1]}{0.1227+[X_1]} + \frac{0.4358[W_2][X_2]}{0.1227+[X_2]} \\ & -0.1800[S_{in}][Y] - 0.3905[X_1][Y] - 0.3905[X_2][Y] \\ & -\frac{0.1155[S_L][Y]}{1.9613+[Y]} - 5.0[T][Y]\end{aligned}$$

$$\frac{d[W_0]}{dt} = \frac{0.5023[S_{in}Y][W^{\oplus}]}{2.9078+[W^{\oplus}]} - 0.2642[W_0^{\ominus}][W_0]$$

$$\frac{d[W_1]}{dt} = \frac{0.2744[X_1Y][W^{\oplus}]}{5.0+[W^{\oplus}]} - 0.2416[W_1^{\ominus}][W_1]$$

$$\frac{d[W_2]}{dt} = \frac{0.2744[X_2Y][W^{\oplus}]}{5.0+[W^{\oplus}]} - 0.2416[W_2^{\ominus}][W_2]$$

$$\frac{d[S_{in}Y]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]}$$

$$\frac{d[X_1Y]}{dt} = \frac{0.4358[W_1][X_1]}{0.1227+[X_1]}$$

$$\frac{d[X_2Y]}{dt} = \frac{0.4358[W_2][X_2]}{0.1227+[X_2]}$$

$$\begin{aligned}\frac{d[W^{\oplus}]}{dt} = {} & -\frac{0.5023[S_{in}Y][W^{\oplus}]}{2.9078+[W^{\oplus}]} - \frac{0.2744[X_1Y][W^{\oplus}]}{5.0+[W^{\oplus}]} - \frac{0.2744[X_2Y][W^{\oplus}]}{5.0+[W^{\oplus}]} \\ & -\frac{0.3[W_1][W^{\oplus}]}{5.0+[W^{\oplus}]} - \frac{0.3[W_2][W^{\oplus}]}{5.0+[W^{\oplus}]} \\ & +\frac{1.0[E^{\oplus}][P]}{1.0+[P]}\end{aligned}$$

$$\begin{aligned}\frac{d[W^{\ominus}]}{dt} = {} & -\frac{0.6[S_{in}Y][W^{\ominus}]}{1.6697+[W^{\ominus}]} - \frac{0.1889[X_1Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - \frac{0.1889[X_2Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} \\ & -\frac{0.3[W_1][W^{\ominus}]}{5.0+[W^{\ominus}]} - \frac{0.3[W_2][W^{\ominus}]}{5.0+[W^{\ominus}]} \\ & +\frac{1.0[E^{\ominus}][P]}{1.0+[P]}\end{aligned}$$

$$\frac{d[W_0^{\ominus}]}{dt} = \frac{0.6[S_{in}Y][W^{\ominus}]}{1.6697+[W^{\ominus}]} - 0.2642[W_0^{\ominus}][W_0]$$

$$\frac{d[W_1^{\ominus}]}{dt} = \frac{0.1889[X_1Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - 0.2416[W_1^{\ominus}][W_1]$$

$$\frac{d[W_2^{\ominus}]}{dt} = \frac{0.1889[X_2Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - 0.2416[W_2^{\ominus}][W_2]$$

$$\frac{d[T]}{dt} = -\frac{0.1155[S_L][T]}{1.9613+[T]} - 5.0[T][Y]$$

$$\frac{d[P]}{dt} = -\frac{1.0[E^{\oplus}][P]}{1.0+[P]} - \frac{1.0[E^{\ominus}][P]}{1.0+[P]}$$

$$\frac{d[E^{\oplus}]}{dt} = \frac{0.1155[S_L][T]}{1.9613+[T]} + \frac{1.0[E^{\oplus}][P]}{1.0+[P]} - 5.0[E^{\ominus}][E^{\oplus}]$$

$$\frac{d[E^{\ominus}]}{dt} = \frac{0.1155[S_L][Y]}{1.9613+[Y]} + \frac{1.0[E^{\ominus}][P]}{1.0+[P]} - 5.0[E^{\ominus}][E^{\oplus}]$$

$$\frac{d[P1^{\oplus}]}{dt} = \frac{0.3[W_1][W^{\oplus}]}{5.0+[W^{\oplus}]}$$

$$\frac{d[P2^{\oplus}]}{dt} = \frac{0.3[W_2][W^{\oplus}]}{5.0+[W^{\oplus}]}$$

$$\frac{d[P1^{\ominus}]}{dt} = \frac{0.3[W_1][W^{\ominus}]}{5.0+[W^{\ominus}]}$$

$$\frac{d[P2^{\ominus}]}{dt} = \frac{0.3[W_2][W^{\ominus}]}{5.0+[W^{\ominus}]}$$

(b) OCN

$$\frac{d[S_{in}]}{dt} = -\frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]} - 0.1800[Y][S_{in}]$$

$$\frac{d[X_1]}{dt} = -\frac{0.4358[W_1][X_1]}{0.1227+[X_1]} - 0.3905[X_1][Y]$$

$$\frac{d[X_2]}{dt} = -\frac{0.4358[W_2][X_2]}{0.1227+[X_2]} - 0.3905[X_2][Y]$$

$$\begin{aligned}\frac{d[Y]}{dt} = {} & \frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]} + \frac{0.4358[W_1][X_1]}{0.1227+[X_1]} + \frac{0.4358[W_2][X_2]}{0.1227+[X_2]} \\ & -0.1800[S_{in}][Y] - 0.3905[X_1][Y] - 0.3905[X_2][Y] \\ & -\frac{3.0[S_F][Y]}{0.1+[Y]}\end{aligned}$$

$$\frac{d[W_0]}{dt} = \frac{0.5023[S_{in}Y][W^{\oplus}]}{2.9078+[W^{\oplus}]} - 0.2642[W_0^{\ominus}][W_0]$$

$$\frac{d[W_1]}{dt} = \frac{0.2744[X_1Y][W^{\oplus}]}{5.0+[W^{\oplus}]} - 0.2416[W_1^{\ominus}][W_1]$$

$$\frac{d[W_2]}{dt} = \frac{0.2744[X_2Y][W^{\oplus}]}{5.0+[W^{\oplus}]} - 0.2416[W_2^{\ominus}][W_2]$$

$$\frac{d[S_{in}Y]}{dt} = \frac{0.5521[W_0][S_{in}]}{2.5336+[S_{in}]}$$

$$\frac{d[X_1Y]}{dt} = \frac{0.4358[W_1][X_1]}{0.1227+[X_1]}$$

$$\frac{d[X_2Y]}{dt} = \frac{0.4358[W_2][X_2]}{0.1227+[X_2]}$$

$$\frac{d[W^{\oplus}]}{dt} = -\frac{0.5023[S_{in}Y][W^{\oplus}]}{2.9078+[W^{\oplus}]} - \frac{0.2744[X_1Y][W^{\oplus}]}{5.0+[W^{\ominus}]} - \frac{0.2744[X_2Y][W^{\oplus}]}{5.0+[W^{\oplus}]}$$

$$\frac{d[W^{\ominus}]}{dt} = -\frac{0.6[S_{in}Y][W^{\ominus}]}{1.6697+[W^{\ominus}]} - \frac{0.1889[X_1Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - \frac{0.1889[X_2Y][W^{\ominus}]}{1.6788+[W^{\ominus}]}$$

$$\frac{d[W_0^{\ominus}]}{dt} = \frac{0.6[S_{in}Y][W^{\ominus}]}{1.6697+[W^{\ominus}]} - 0.2642[W_0][W_0^{\ominus}]$$

$$\frac{d[W_1^{\ominus}]}{dt} = \frac{0.1889[X_1Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - 0.2416[W_1^{\ominus}][W_1]$$

$$\frac{d[W_2^{\ominus}]}{dt} = \frac{0.1889[X_2Y][W^{\ominus}]}{1.6788+[W^{\ominus}]} - 0.2416[W_2][W_2^{\ominus}]$$

$$\frac{d[F]}{dt} = \frac{3.0[S_F][Y]}{0.1+[Y]}$$

2 Interaction Series

Table 4: The interaction series that represents an experimental protocol of FCNN learning for the target binary function $f(x_1,x_2)$. The random weight setting at time 0 is performed only initially, the rest of injections and assignments defined at time 100, 140, and 200 are repeated with periodicity 500 each learning iteration. The learning parameters are defined as follows: penalty signal concentration (learning rate) $\alpha = 1$, annealing rate $k = 0.0008$, and threshold concentration $\theta = 0.6$.

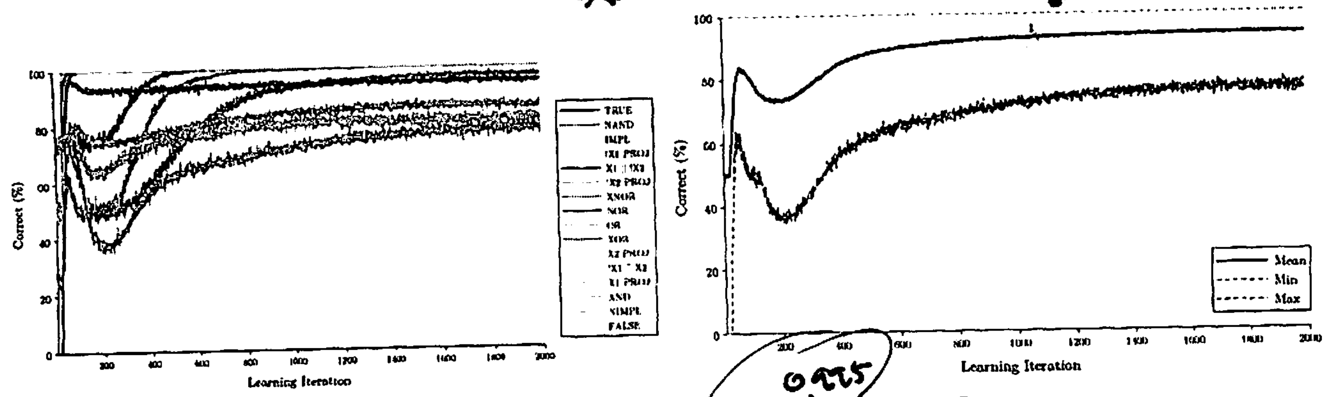
Time	Injections/Assignments
0	pick $[W_0] \in (0.5, 1.5)$
	pick $[W_1] \in (0.5, 1.5)$
	pick $[W_2] \in (0.5, 1.5)$
100	pick $x_1 \in \{0, 1\}$
	pick $x_2 \in \{0, 1\}$
	$[S'_{in}] = .5$
	$[X'_1] = x_1$
	$[X'_2] = x_2$
	$[S_{in}Y] = [X_1Y] = [X_2Y] = 0$
	$[E^{\oplus}] = [E^{\ominus}] = 0$
	$[S_L] = 0$
140	$[S'_F] = 1$
200	$incorrect = [Y] > \theta \neq f(x_1,x_2)$
	$\alpha = (1-k)\alpha$
	$[T] = \theta \quad if\ incorrect$
	$[P] = \alpha \quad if\ incorrect$
	$[S_L] = 1 \quad if\ incorrect$

Name	f(1,1)	f(1,0)	f(0,1)	f(0,0)
FALSE	0	0	0	0
NOR	0	0	0	1
NCIMPL	0	0	1	0
NOT X1	0	0	1	1
NIMPL	0	1	0	0
NOT X2	0	1	0	1
XOR	0	1	1	0
NAND	0	1	1	1
AND	1	0	0	0
NXOR	1	0	0	1
PROJ X2	1	0	1	0
IMPL	1	0	1	1
PROJ X1	1	1	0	0
CIMPL	1	1	0	1
OR	1	1	1	0
TRUE	1	1	1	1

Figure 1: All 2-input logic functions with associated truth (output) tables.

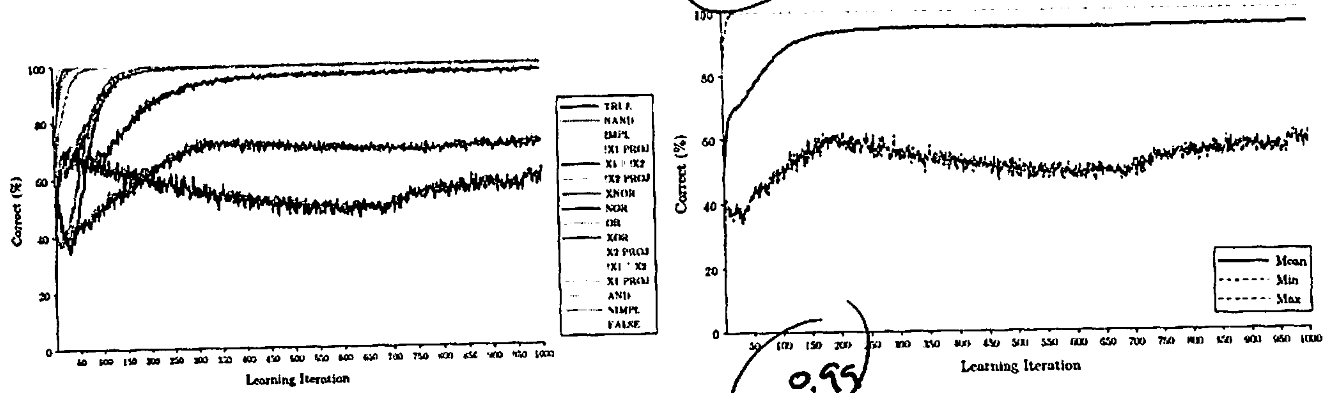
3 Performance of Alternative Models

Description ?



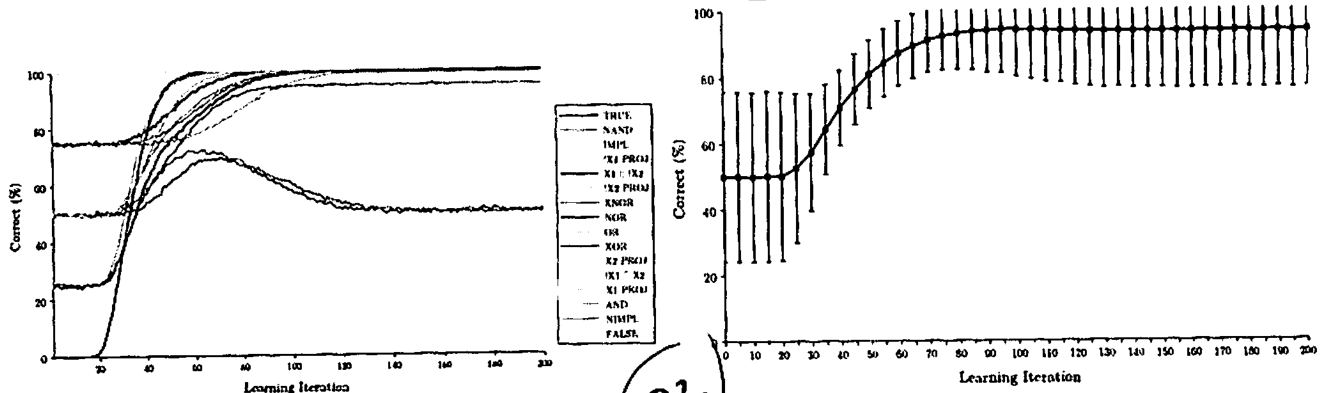
0.995

(a) FCNN SASP + 2xAASP (P = 1.0, anneal k = .995): mean 93.5



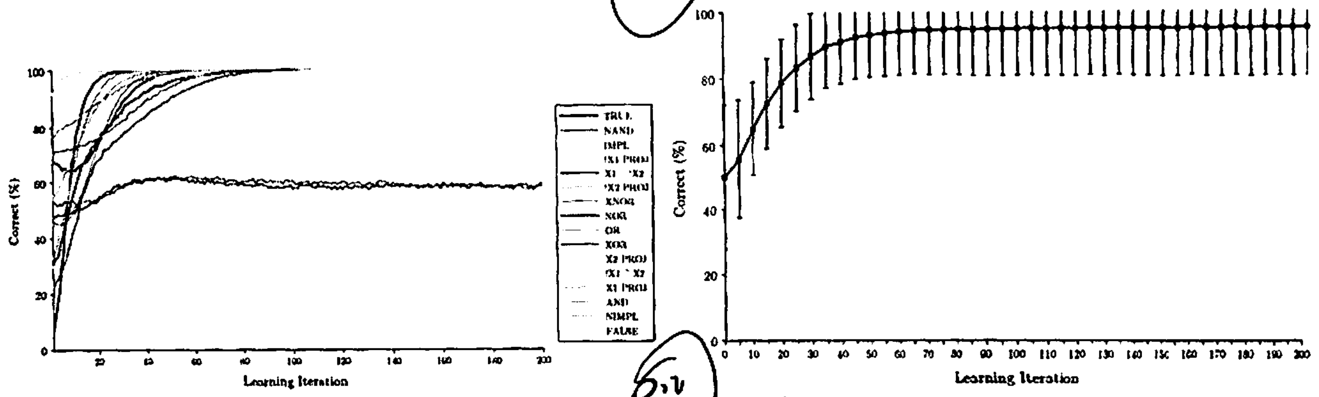
0.99

(b) FCNN TASP + 2xAASP (P = 1.0, anneal k = .99): mean 95.4



0.2

(c) SASP (P = .2): mean 93.4



0.2

(d) TASP (P = .2): mean 94.8