Nanoscale

ROYAL SOCIETY OF CHEMISTRY

MINIREVIEW

View Article Online
View Journal | View Issue

Cite this: *Nanoscale*, 2023, **15**, 7676

A survey on molecular-scale learning systems with relevance to DNA computing

Rajiv Teja Nagipogu, *† Daniel Fu *† and John H. Reif

DNA computing has emerged as a promising alternative to achieve programmable behaviors in chemistry by repurposing the nucleic acid molecules into chemical hardware upon which synthetic chemical programs can be executed. These chemical programs are capable of simulating diverse behaviors, including boolean logic computation, oscillations, and nanorobotics. Chemical environments such as the cell are marked by uncertainty and are prone to random fluctuations. For this reason, potential DNA-based molecular devices that aim to be deployed into such environments should be capable of adapting to the stochasticity inherent in them. In keeping with this goal, a new subfield has emerged within DNA computing, focusing on developing approaches that embed learning and inference into chemical reaction systems. If realized in biochemical contexts, such molecular machines can engender novel applications in fields such as biotechnology, synthetic biology, and medicine. Therefore, it would be beneficial to review how different ideas were conceived, how the progress has been so far, and what the emerging ideas are in this nascent field of 'molecular-scale learning'.

Received 6th November 2022,
Accepted 2nd April 2023

DOI: 10.1039/d2nr06202j

rsc.li/nanoscale

1. Introduction

Learning and adaptation occur naturally in living systems and help them to survive in ever-changing environments. It is not necessary for these systems to be complex, as evidence suggests that simple biochemical circuits evolved by microorganisms allowed them to search for food, avoid predators, and adapt to the uncertainty inherent in chemical environments.[1] For example, Hennessey *et al.*[2] demonstrated that the single-celled ciliate Paramoecium could be classically conditioned by training it to expect an electric-shock stimulus whenever it encountered a vibrational stimulus. During the test phase, the Paramoecium actively avoided the regions with a vibrational stimulus, evidence that it expects an electric shock if it enters that region. Along similar lines, Fernando *et al.*[3] designed a simplified version of the gene regulatory network that can be programmed to associate any pair of signals taken from a predefined set. Understanding the principles underlying such behaviors could enable us to comprehend, repair, and maybe even build systems at the complexity of life itself.

In this regard, we believe that the field of molecular computing is well-poised to drive innovation in the following ways: (i) it can be a window through which we can glean the computational capabilities of naturally occurring biochemical systems, and (ii) build synthetic chemical networks of increasing complexity to mimic or interact with the biochemical circuits inside living systems. While the former is a theoretical pursuit, the latter takes an engineering approach asking the question - *what can we build*? We can draw a close parallel to the field of artificial intelligence, where a faction of researchers (*e.g.*, in neuroscience) study the inner processes of the brain and their role in manifesting intelligence, whereas another faction (*e.g.*, deep learning) focuses solely on building models that increasingly approach the function of the brain without deliberating on its underlying processes.

Historically, there have been various attempts towards modeling learning at the molecular scale. However, to restrict the scope of this review, we focus only on the works that have direct relevance to a subfield of molecular computing commonly known as DNA computing.

DNA computing, as the name suggests, uses the chemical reactions between biomolecules such as DNA to simulate computation in chemistry. Compared to tracing the history of traditional machine learning, the work in molecular-scale learning still appears scattered, but its breadth of coverage of machine learning topics is becoming increasingly thorough (Fig. 1). Several reasons exist as to why DNA computing is well-poised to drive progress in molecular computing: (i) DNA has predictable behavior through well-understood Watson–Crick base pairing rules of nucleotides, (ii) increasingly complex molecular machinery is being designed and implemented due to an interdisciplinary interest in the area from biologists, chemists, physicists, and computer scientists, and (iii) widely

Department of Computer Science, Duke University, Durham, NC, USA.
E-mail: rn118@cs.duke.edu, dfu@cs.duke.edu
† These authors contributed equally to this work.

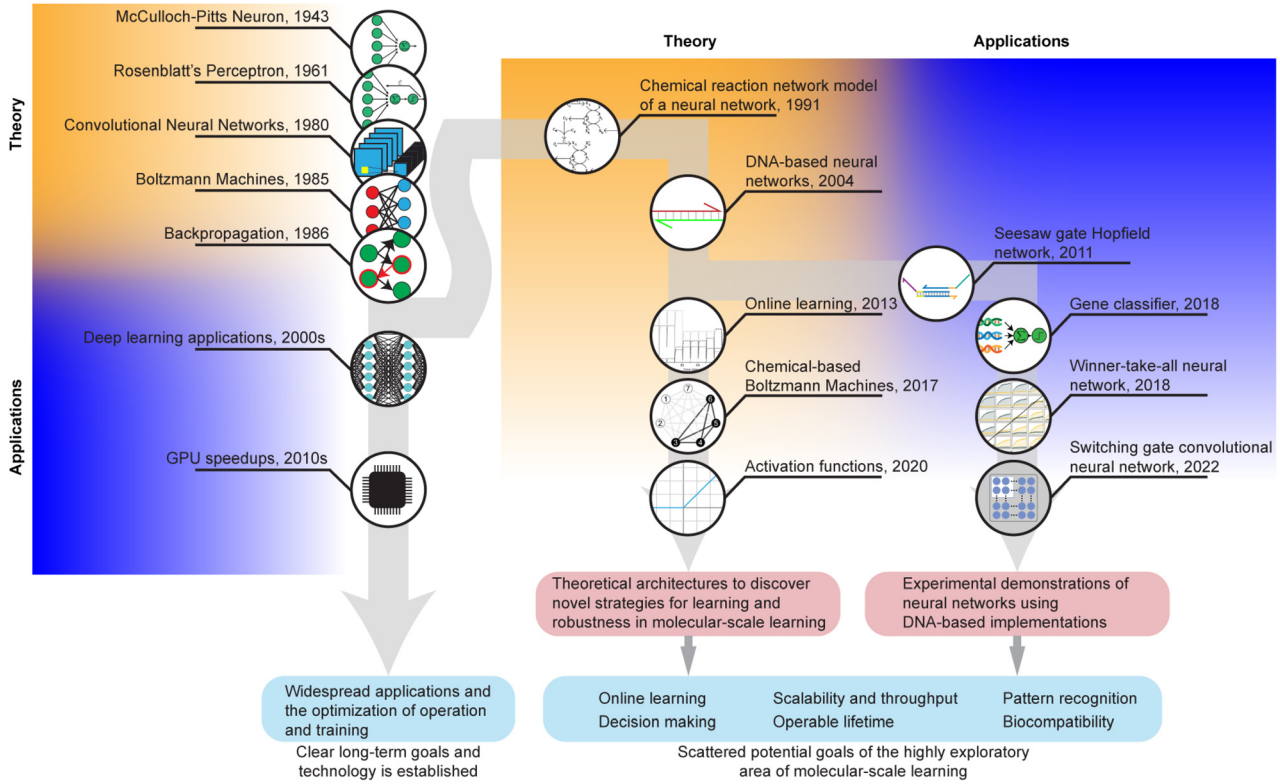

Fig. 1 Summary of milestone concepts in artificial intelligence and its adaptations into molecular computing. It is straightforward, in hindsight, to trace the development of ANNs and notice the presence of past contributions in modern technologies. Applications have followed as a consequence of the maturation of theory. On the other hand, no such direct path can be traced or forecasted for the same developments in molecular computing. Practical demonstrations and theoretical constructions often do not overlap but proceed simultaneously. Moreover, each subsequent study introduces different perspectives on the development of the field. It may have the advantage of adapting techniques that are already very established, but molecular computing itself has a diverse toolbox. Whether the existing technologies will consolidate in operation or application remains to be discovered. This figure has been adapted from: Hjelmfelt *et al.*[4] with permission from National Academy of Sciences, copyright 1991; Cherry *et al.*[5] with permission from Springer Nature, copyright 2018; Xiong *et al.*[6] with permission from Springer Nature, copyright 2022; Lopez *et al.*[7] with permission from Springer Nature, copyright 2018; Banda *et al.*[8] with permission from MIT Press, copyright 2013; Vasic *et al.*[9] with permission from arXiv,[9] copyright 2020; Poole *et al.*[10] with permission from Springer, copyright 2017.

available synthesis and chemical functionalization of custom DNA oligonucleotides can facilitate their interaction with natural biological systems.[23–28] The properties of DNA that are simultaneously programmable and biocompatible have encouraged broad research into biosensing,[29] drug delivery,[26] and diagnostics,[30,31] and the addition of machine learning would transform how those applications could interact with their molecular-input data.

However, despite the promise shown by the field, the number of practical implementations of neural network-like computation has been noticeably low, owing to the scale and complexity of implementing such systems. This number reduces further when we consider implementations that include learning. For this reason, alongside the works that include practical implementation of neural network-like learning and computation, we also include results that are primarily theoretical and/or simulations but hint at a future DNA computing-based implementation. Table 1 briefly introduces several works and provides a high-level overview showing the progress of topics in molecular-scale learning.

The review is structured as follows: in sections 2 and 3, we provide a brief background into the fields of artificial intelligence, molecular computing, chemical reaction networks, and their association with DNA computing. In section 4, we survey the works from the DNA computing community that implement neural network-like computation and learning at the molecular level, presented in chronological order. Finally, in section 5, we discuss the challenges faced by DNA-based implementations concerning scalability, robustness, and responsiveness and offer perspectives for future implementations.

2. Background

2.1 A brief historical overview of artificial intelligence

The field of artificial intelligence (AI) is concerned with building agents that embody capabilities to learn, make intelligent decisions, and survive in uncertain conditions. Although the field started as an endeavor to understand and replicate the

Table 1 Chronological summary of the works detailed in this review. The first column provides a short description of the work. The second column is the year in which the work was published. The third column represents how the results of the work were generated, *i.e.*, through simulations, *in vitro* experiments, *etc*. The fourth column shows whether the work tries to accomplish learning in chemistry. The fifth column is a list of notable facts regarding each work

Work	Year	Mode	Learning?	Results/miscellaneous
CRNs for McCulloch–Pitts neuron and linear boolean logic gates[4]	1991	ODE simulators	No	Computation of AND, OR functions in chemistry
RNase & RNAP-based DNA transcriptional switches[11]	2004	Simulations of enzymatic *in vitro* DNA computing	No	Feedforward circuits, Hopfield memory, and WTA circuits
Neural network-like computation with seesaw gates[12]	2011	*In vitro* implementation w/DNA computing	No	4-Bit Hopfield associative memories
Enzymatic winner-take-all circuit[13]	2013	*In vitro* implementation w/enzymatic DNA computing	No	4-Bit inputs evaluated by 23 strand DNA-based WTA implementation *via* PEN toolbox
Two designs for a boolean chemical perceptron[8]	2013	Artificial chemistry	Yes	21 species, 34 reactions 13 species, 16 reactions
Analog asymmetric signal perceptron[14]	2014	Artificial chemistry	Yes	17 species, 18 reactions
Perceptron learning algorithm using DNAzymes[15]	2014	Simulations w/DNA computing	Yes	Positive weights only, unstable when weights closer to zero
Perceptron learning algorithm using DNAzymes[16]	2016	Simulations w/DNA computing	Yes	278 different kinds of gates
Feedforward chemical neural network, a network of AAS perceptrons[17]	2017	ODE simulators	Yes	Linearly inseparable functions, with a hidden layer
Boltzmann machines, expectation–maximization algorithm, and Baum–Welch algorithm implemented using stochastic CRNs[10,18,19]	2017–2019	Simulations	No	More suited to microscopic environments
Multi-gene linear classifier[7]	2018	*In vitro* implementation w/DNA computing	No	Measuring gene expression from RNA for early cancer diagnostics and differentiating viral and bacterial infections
Pattern recognition using winner-take-all computation[5]	2018	*In vitro* implementation w/DNA computing	No	Extendable design, no learning required, 100-bit pattern recognition
Binary-weight ReLU network[9]	2020	Simulations	No	Implementation of ReLU activation function using rate-independent CRNs
Theoretical foundations for CRN implementations of neural networks[20]	2021	Simulations	No (but hints at an extension in the future work)	Neural network with a hidden layer using a smoothed version of ReLU
Pattern recognition using nucleation kinetics of self-assembly[21]	2022	*In vitro* implementations w/DNA computing	No	917 different tile species
CRN and DNA-based architectures for a spiking neuron[22]	2022	VisualDSD simulator	Yes	Associative memory between 5 stimuli, memory decay
DNA switching gate ConvNet[6]	2022	*In vitro* implementation w/DNA computing	No	144-Bit pattern recognition, two-layer implementation

function of the human brain as well as an inquiry into the philosophy of mind, in the past few decades, it has been steadily moving away from being a philosophical inquiry to being an engineering pursuit, significantly towards engineering simplified abstractions of brain function known as neural networks. Further, the advent of specialized hardware with massively parallel processing capabilities (*e.g.*, GPU, TPU) enabled researchers to build increasingly large neural networks, a subject of study commonly known today as "deep learning". The capacity of these networks to learn and perform a huge variety of tasks has resulted in their widespread deployment into every corner of science, technology, and industry.[32–37] We refer the reader to other surveys[38–43] for a more detailed discussion on the history and applications of deep learning.

2.2 Perceptron

A traditional neuron can exist in one of the two states: (i) firing, when the magnitude of its input is greater than a threshold, and (ii) silent, otherwise. A perceptron (Fig. 2b) is an abstract mathematical model of the neuron. It is a network of nodes arranged into two layers with weighted edges running from the first layer to the second layer with no connections between nodes of the same layer. The *modus operandi* of a perceptron can be divided into two phases: (i) the forward phase and (ii) the learning phase. In the forward phases, the perceptron reads its inputs, calculates a weighted sum, and if the sum is greater than a threshold, it fires and stays silent otherwise.

A perceptron is different from an algebraic network in that it can learn. This learning happens through a simplified analog of an algorithm based on gradient descent known as backpropagation.[44] In the learning phase, the model is provided with a set of input-target pairs $(x_1, d_1),\ldots,(x_n, d_n)$ *a.k.a.* the training set. It then iterates over these pairs and calculates the weighted sum of the input y_i. The error e_i, *i.e.*, the discrepancy in the output predicted by the model and the desired

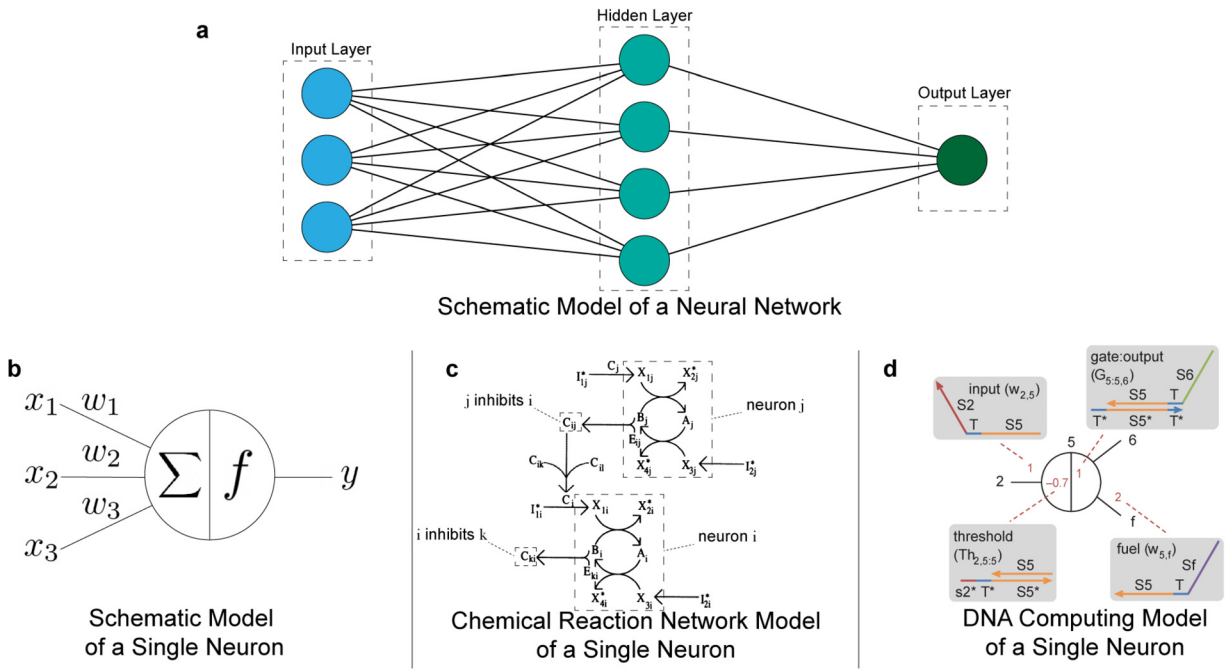

Fig. 2 The neural network paradigm of learning as it follows a transition from digital representations into a DNA-based medium *via* chemical reaction networks. (a) A simple ANN has an input layer, an output layer, and often multiple hidden processing layers between the terminal layers. (b) The longstanding model of a single node is a perceptron with inputs x_i and corresponding weights w_i, integration function $\sum$, thresholding function f, and output y. (c) The signal propagation between two neurons translated into a chemical reaction network representation. This figure has been adapted from Hjelmfelt *et al.*[4] with permission from National Academy of Sciences, copyright 1991. (d) A DNA-based neuron constructed based upon the seesaw gate motif. This figure has been adapted from permission from Qian *et al.*[12] with permission from Springer Nature, copyright 2011.

output (d_i), are compared. The model weights are then adjusted using the perceptron weight update rule: $w_i = w_i + \alpha(d - y)x_i$, where α represents the learning rate which signifies the adaptation strength of the network. The weights are initialized randomly, and the updates will be performed over the entire training set until all the weights stabilize. The resultant network can then be used to generate outputs for the unseen inputs. Several models of a perceptron exist in the literature, among which the ones by McCulloch–Pitts[45] and Rosenblatt[46] are prominent, and the large networks assembled from these perceptrons as nodes are today referred to as artificial neural networks (ANNs).

3 DNA computing

DNA computing is a computational paradigm that aims to perform computation at a molecular scale by utilizing the reactions between biomolecules such as DNA. Ontologically, it is a subfield of molecular computing which refers broadly to the techniques that utilize molecules, such as enzymes, proteins, *etc.*, as substrates for computation. The field traces its origins back to the infamous experiment conducted by Leonard Adleman[81] in 1994, where he demonstrated that it is possible to solve the Hamiltonian path problem, a well-documented NP-hard problem in computer science, using the parallelism inherent in chemical reactions, particularly DNA hybridization reactions. He devised a way to encode the nodes and edges in the original graph into DNA molecules so that in a well-mixed solution, these molecules will form linear chains of valid paths owing to the selective hybridization properties of DNA molecules. The initial result attracted the interest of the research community, and this new paradigm was heralded to solve hard combinatorial problems in computer science. However, it was soon realized that this mode of computing scales poorly, as the circuit size would be exponential in the problem size. While this could have objectively been the end of the field, researchers realized that this paradigm could perform computation in biological contexts such as the cell where traditional hardware is currently ineffective. As the field progressed, it was shown that a class of DNA reactions commonly known as DNA strand displacement (DSD) is capable of universal computation,[49] which revitalized the position of the field as an alternative form of computing. A more detailed survey on the evolution of DNA computing, techniques, methodologies, and promising areas of application can be found here.[50,51] We provide a graphical summary of popular DNA computing techniques in Fig. 3.

When we look towards the chemical medium for implementing models of learning (Fig. 2), we must be cognizant of the limitations that may arise due to several factors, such as the physical properties of the materials involved (*e.g.*, DNA in DNA computing), possible slower reaction times, erroneous reactions and results, *etc*. However, seeking alternative models

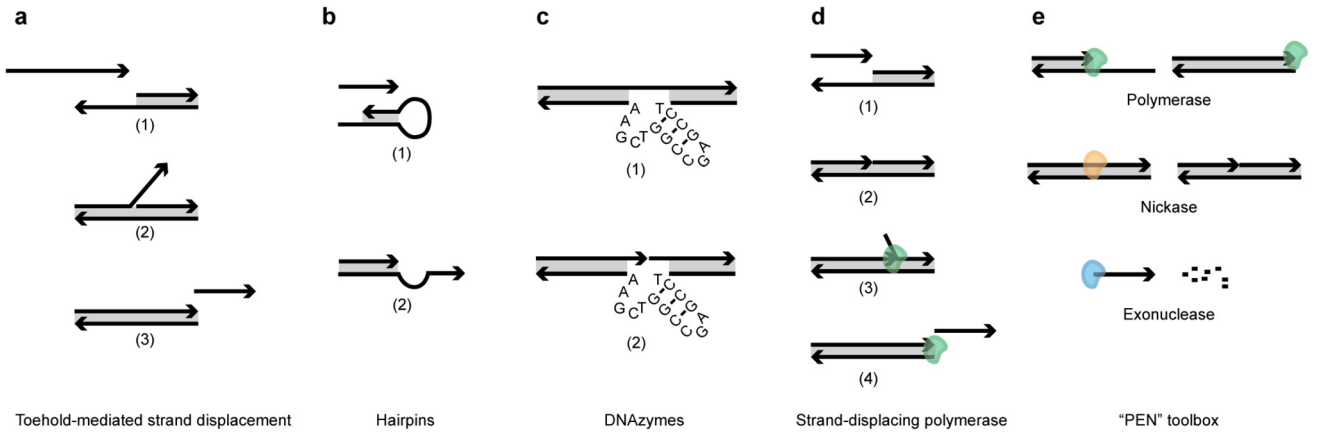

Fig. 3 Common mechanisms for DNA computing. (a) Toehold-mediated strand displacement is a fundamental technique that belies most DNA computing. (b) Hairpins are a secondary structure of DNA strands. They contain a "stem" and a "loop". Domains within the loop are often less reactive until the stem is opened *via* toehold-mediated strand displacement. (c) DNAzymes are artificially discovered secondary structures of DNA strands that can cleave a complementary nucleic acid strand. The example shown is an 8–17 DNAzyme.[47] (d) Strand-displacing polymerase is a subclass of polymerase that simultaneously displaces the incumbent strand. (e) The PEN toolbox[48] contains polymerase, exonuclease, and nickase enzymes to facilitate the production and annihilation of DNA strands.

of computation has always been an eager pursuit. As we continue our discussion in this review, we should keep in mind that the aim of these devices isn't to replace conventional silicon-based hardware. Rather, it is to embed computation where silicon-based systems cannot currently go (*e.g.*, a cell).

3.1 Chemical reaction networks

A chemical reaction network (CRN) is a mathematical formalism to abstract and study the dynamics of a system of chemical reactions. It consists of a finite set of chemical reactions of the form shown in eqn (1), where X_i and Y_j represent the reactant and product species, respectively, while α_i and β_j represent their stoichiometric coefficients. N_r and N_p represent the number of distinct reactant and product species.

$$\sum_{i=1}^{N_r} \alpha_i X_i \xrightarrow{k_r} \sum_{j=1}^{N_p} \beta_j Y_j \tag{1}$$

A constituent reaction of a CRN. X_i represents the reactant species and Y_j represents the product species. α_i and β_j represent the stoichiometric coefficients of the reactants and products.

The temporal dynamics of the system can be estimated using the rules of mass action kinetics, resulting in a system of ordinary differential equations (ODEs). CRNs can be designed to exhibit complex behaviors, such as controllers, oscillators, and chaotic systems. As a CRN maps directly onto a system of ODEs, its temporal dynamics can be modeled using traditional ODE simulators[52] in a straightforward manner. DNA has become a popular substrate for designing CRNs as it offers ways to synthesize orthogonal species that result in rigidly defined reaction pathways with modulable reaction rates.

4. Survey of molecular computing approaches to learning

4.1 Neural network-like computation to simulate logic functions

One of the earliest attempts at neural network-like computation in a chemical medium is by Hjelmfelt *et al.*[4] Here, the authors devise a reversible enzyme reaction system to simulate the dynamics of a McCulloch–Pitts neuron.[45] This neuron has two modes of operation: (i) firing and (ii) quiescent, depending on the concentration of its output relative to a threshold. The neuron fires if the output concentration is greater than a threshold and stays quiescent otherwise.

$$\begin{aligned} I_{1i}^{*} + C_i &\rightleftharpoons X_{1i} + C_i \\ X_{1i} + B_i &\rightleftharpoons X_{2i}^{*} + A_i \\ X_{3i} + A_i &\rightleftharpoons X_{4i}^{*} + B_i \\ X_{3i} &\rightleftharpoons I_{2i}^{*} \end{aligned} \tag{2}$$

A CRN that represents a chemical implementation of a single McCulloch–Pitts neuron by Hjelmfelt *et al.*[4] The chemical species C_i represents the total input to the neuron, and the chemical species A_i represents its output. B_i is a species complementary to A_i in that their sum adds up to 1. Further, the concentrations of the species marked using an asterisk are kept constant throughout the operation of the circuit.

The CRN in eqn (2) simulates the dynamics of a single McCulloch–Pitts neuron. Computation is enabled through the dynamics of chemical reactions. The chemical species C_i and A_i represent the input and output of the neuron. B_i is a complementary species to A_i such that their concentrations have a constant sum. Further, the species marked with an * are maintained at a constant concentration throughout. The CRN operates in such a way that when the concentration of C_i grows above a predefined threshold, a steep rise in the concentration

of A_i is observed, indicating that the neuron has fired. In addition, communication between different neurons is enabled by allowing the output species of the j^{th} neuron (A_j) to act as an input to the i^{th} neuron (C_{ij}). Using this framework, they prototyped networks that compute two-input linearly separable boolean logic functions such as AND and OR.

4.2 Neural network computation by *in vitro* transcriptional circuits

Kim *et al.*[11] explored the equivalence between neural network computation and genetic regulatory networks. They proposed a simplified biochemical analog of a gene regulatory network using the actions of the enzymes RNA polymerase (RNAP) and ribonuclease (RNase) as reaction primitives. The basic unit of their circuit is a DNA transcriptional switch, a device that takes in an input and releases an output. The switch embodies three important characteristics: (i) the presence of input and output domains, (ii) transcribing efficiently when in an activated state, and (iii) transcribing poorly when in a deactivated state. The switch remains deactivated when it is in its native state and becomes active when an activator molecule is bound to it, similar to the functioning of a gene.

The reactions at a switch are shown in Fig. 4a. D is the deactivated (native) state of the switch, referred to as the OFF state, and DA is the activated state, referred to as the ON state. A (red) represents the activator strand (RNA) that can convert the switch from OFF state to ON state by binding to it, and I (green) represents the inhibitor strand (complementary to A) that can bring back the switch to OFF state. The enzymes RNAP and RNase drive changes in the circuit by modulating the concentrations of the RNA signal strands A and I.

First, they established that the ODE system of their model is equivalent to the ODE system dynamics of the Hopfield neural network model.[53] Further, by developing a communication protocol among the switches (much similar to that in Hjelmfelt *et al.*,[4] they provided designs for implementing larger circuits such as a boolean feedforward network, associative memories, and a winner-take-all network (Fig. 4(b–d)).

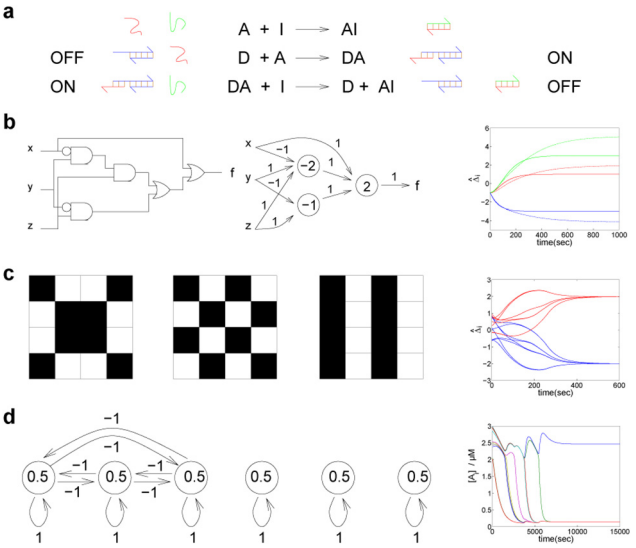

Fig. 4 An enzyme-based neural network-like circuit. (a) Binding reactions that implement a DNA-based transcriptional switch. (b) A feedforward construction that computes $f(x, y, z) = \bar{x}yz + \bar{y}z + x$. The graph shows an instance run where x = true, y = false, z = true. (c) A Hopfield associative memory for 16-bit patterns. The graph shows an example of the network categorizing the black or white columns of the third pattern, with red lines representing even-numbered columns and blue lines representing odd-numbered columns. (d) A 3-bit WTA network. The left diagram shows the intended relationships between each bit. The middle diagram shows its implementation, which defines the inhibitory conditions implicitly as the competitive consumption of RNAP, which catalyzes each independent reaction. The graph shows an example of a 10-bit input, where only k = 1 winners are desired. This figure has been reproduced from Kim *et al.*[11] with permission from MIT Press, copyright 2004.

4.3 Neural network computation using DNA seesaw gates

So far, the chemical systems proposed are primarily theoretical due to the lack of a programmable substrate. DNA computing alleviates this deficit to some extent. Qian *et al.*[12] implemented a linear threshold gate using DSD reactions (Fig. 3) at a special kind of auxiliary gate complexes known as seesaw gates. A gate refers to a partial DNA duplex with nucleation centers that facilitate strand displacement. The resultant DNA circuit is comprised of two components: (i) seesaw gates and (ii) signal strands. A seesaw gate[54] is a partially hybridized double-stranded DNA with the longer base strand hybridized to a shorter incumbent strand. The base strand contains a longer domain flanked on either side by toeholds that hybridize to single-stranded substrates (signals or fuel) in an XOR manner – one toehold stays free when the other is bound. A signal strand is a single-stranded DNA molecule that consists of a short toehold domain flanked on either side by two domains named the left recognition domain (LRD) and the right recognition domain (RRD), each representing the upstream gate the strand was released from and the downstream gate the strand is incident upon. In the basic operation of a seesaw gate, an input signal strand impinges on the free toehold, displacing the incumbent strand and freeing the previously bound toehold. A fuel strand then binds to this toehold to displace the input signal. In this way, the input signal effectively catalyzes the formation of the output.

The binary linear threshold gate implemented by Qian *et al.*[12] involves three primary operations: (i) weight multiplication ($w_i x_i$), (ii) integration ($\sum w_i x_i$), and (iii) thresholding ($\sum w_i x_i \geq$ th), simulated through custom-designed seesaw gates. At the multiplication gate, the presence of input triggers a release of output strands stoichiometrically equivalent to the number of gates, simulating "binary weight multiplication". Two reactions occur at this seesaw gate: (i) the input strand impinges onto the free toehold and displaces the output strand, and (ii) the fuel strand impinges on the other (now free) toehold and displaces the input strand. In this way, all the multiplication gates would be exhausted, and outputs strands equivalent to the weight would be released. Integration

gates take in the outputs from the multiplication gate and release a common output strand simulating the sum operation. The thresholding gates then absorb a stoichiometrically equal amount of these common output strands, resulting in the thresholding operation. The output strands that remain after thresholding act as the final output of the linear threshold gate.

Using this framework, the authors demonstrated a 4-bit Hopfield memory that was able to classify among four 4-bit patterns (Fig. 5c & d). The authors opined that despite its simplicity, the binary linear threshold gate could find applications in medical diagnostics in tasks such as classifying cancer cells using mRNA signals.

This work signifies a significant leap in the demonstrations of neural network-like computation in chemistry using DNA molecules. As we continue our survey, we will review additional works in the literature that demonstrate improvements in size, complexity, and utility.

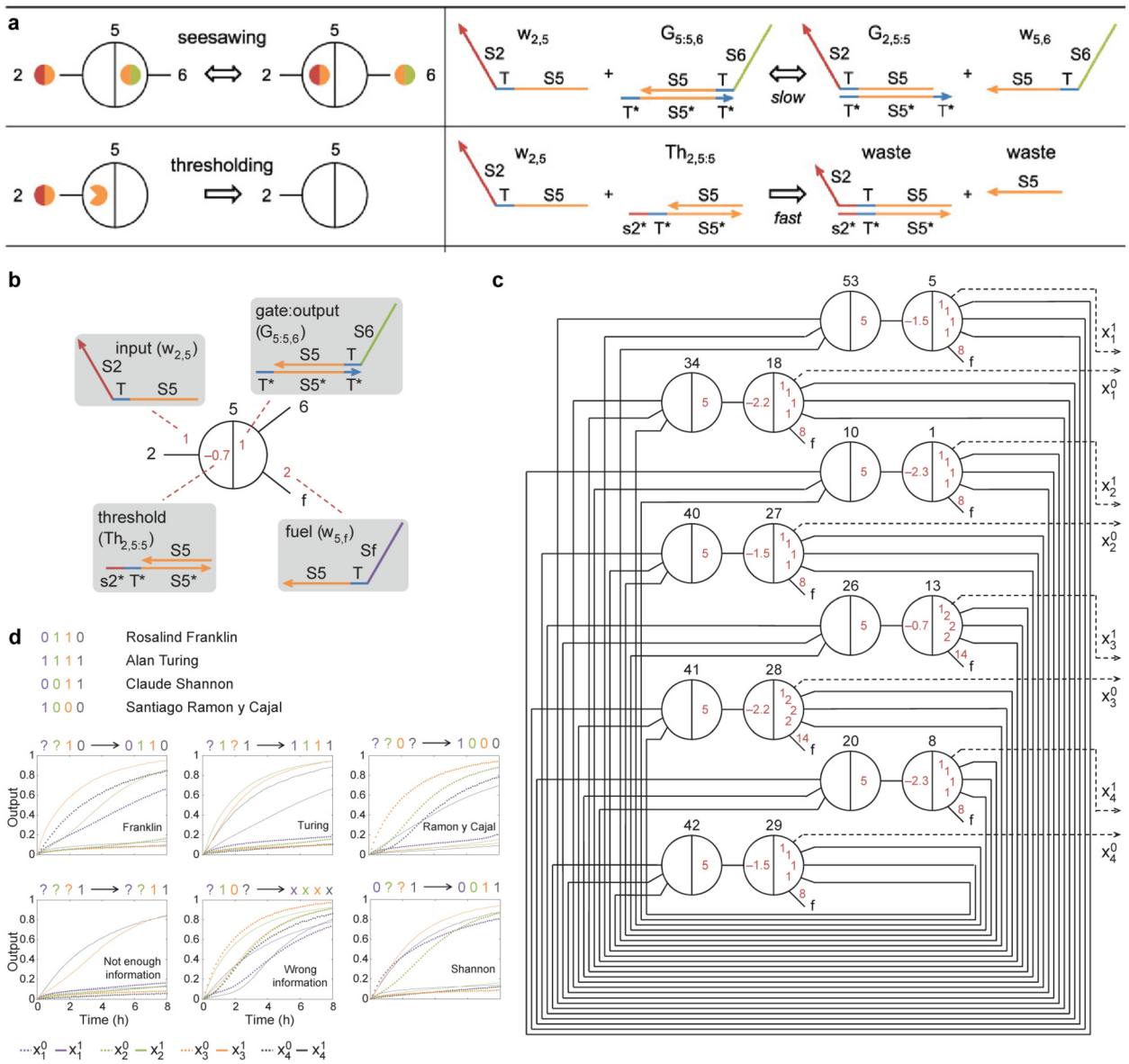

Fig. 5 Brief overview of neural networks based on the DNA seesaw mechanism. (a) Operation of a seesaw gate circuit is based on seesawing and thresholding operations implemented as shown. This figure has been adapted from Qian *et al.*[55] with permission from AAAS, copyright 2011. (b) The presence of strands and their concentrations can be expressed in a graphical abstraction as a single node. This figure has been adapted from Qian *et al.*[55] with permission from AAAS, copyright 2011. (c) Example network showing the node connection configuration to implement a 4-bit Hopfield associative memory using DNA seesaw gates. (d) The neural network was trained *in silico* to recognize four different scientists, each assigned a 4-bit label, even when given incomplete information. This figure has been adapted from Qian *et al.*[12] with permission from Springer Nature, copyright 2011.

4.4 Online learning in a chemical perceptron

It can be noticed that the DNA circuit devised by Qian *et al.*[12] to simulate a binary threshold gate is a one-shot device, evident from the fact that all the multiplication gates are used up in one forward pass of the input. Future DNA systems that intend to be deployed in chemical environments should embody the capacity to be active for extended durations, learn from their surroundings, and make decisions autonomously. Banda *et al.*[8] tackled this problem by designing a CRN that emulates perceptron-like behavior using the artificial chemistry (AC) framework.[56] AC represents boolean values using two categorical chemical species, S^0 and S^1 (representing 0 and 1, respectively), whereas the real values are represented using the concentrations of the corresponding chemical species. Using this framework, they put forward designs for two qualitatively distinct implementations of a formal perceptron: (i) the weight-loop perceptron and (ii) the weight-race perceptron.

The operation of the weight-loop perceptron adheres closely to that of a formal perceptron. First, the input-weight multiplication is achieved by allowing the input species X_i to catalyze the transformation of the weight species W_i^p into the output species Y^p (p: = parity $\in$ 1, 0). Then, the output species Y^1 and Y^0 resulting from different input-weight integrations annihilate each other in a fast reaction until one is exhausted (eqn (3)). The learning phase begins with the introduction of the target species D^p. The target species then releases the weight-changer species if its parity differs from the output species and stays dormant otherwise. These weight-changer species react with the weight species adjusting their magnitude according to the perceptron weight update rule.

One of the drawbacks of the weight-loop perceptron is that it translates the digital computations directly into chemistry, leading to cumbersome implementations. The weight-race perceptron (Fig. 6b) aims to simplify this design to pave the way for an easier chemical implementation. Here, the roles of the input and weight species are switched, *i.e.*, the weight species now catalyzes the transformation of the input species (if present) into the output species. For example, the species $W_1^{\oplus}$ and $W_1^{\ominus}$ could convert the positive species $X_1{}^1$ into Y^1 or Y^0, respectively, while being unreactive on the negative species $X_1{}^0$. However, the bias species W_0 can act on both the inputs X_1 and X_2, creating a weight race, *i.e.*, the weights will have a disproportionate impact on the output formed as multiple weights can now result in the same output. To ensure that the weight race is fair, they adjust the rate constants and introduce new decay reactions into the system. We direct the reader to the original work[8] for a more detailed discussion of the discussed weight race and the reactions involved.

This work marked the first-ever attempt at "online learning" in a chemical medium by introducing two classes of chemical perceptrons. The operation of the weight-loop perceptron uses 21 species and 34 reactions and involves shoehorning digital computations into chemical reaction cascades, leading to a cumbersome implementation. In contrast, the weight-race perceptron adopts a more chemistry-friendly approach by allowing the weights to catalyze the transformation of inputs to outputs reducing the network size to 14 species and 30 reactions. Both the perceptrons were trained to recognize the patterns in the NAND function.

4.5 Online learning in analog asymmetric signal perceptron (AASP)

The chemical perceptrons introduced by Banda *et al.*[8] could only learn linearly separable boolean functions. On the other hand, many naturally occurring processes have nonlinear dependencies between their inputs and outputs. Banda *et al.*[14] extended the design of weight-race perceptron to form a new model named the analog asymmetric signal perceptron (AASP) that can simulate nonlinear functions of the form $y = \varphi(w_1x_1 + w_2x_2 + w_0)$, where φ is a nonlinear activation function such as the sigmoid. The values of the inputs and weights are represented using the concentrations of the corresponding input and weight species X_i & W_i. Nonlinear input-weight integration was achieved through a system of two competing reactions: (i) weight W_i catalyzes the transformation of the input species X_i into output species Y and (ii) input X_i reacts annihilatively with the output species Y to form a nonreactive waste $X_i + Y \rightarrow \lambda$. Learning in the circuit is triggered by the addition of the target species $\hat{Y}$ and follows the perceptron weight update rule: $\Delta w_i = \alpha(\hat{y} - y)x_i$. First, the output and target species annihilatively, simulating the subtraction $(\hat{y} - y)$ where the excess of one results in the formation of the weight changer species $W^{\oplus}$ or $W^{\ominus}$ according to the weight update rule. These weight-changer species then react with the weight species and updates their values.

The AASP, the weight-loop perceptron, and the weight-race perceptron described above established explicit models capable of online learning chemistry. The corresponding works also hint at future DNA implementations. They estimate that the weight-race perceptron might require 40–50 unique strands, a quantity within the demonstrated state-of-the-art circuit size (between 100–200 strands) at the time.

$$\begin{gathered} W_i \xrightarrow{X_i=S^1} \bar{W}_i + Y^1 \\ W_i \xrightarrow{X_i=S^0} \bar{W}_i + Y^0 \\ \bar{W}_i + E \rightarrow W_i \\ Y^1 + Y^0 \rightarrow \phi \end{gathered} \tag{3}$$

CRNs for the weight-loop perceptron introduced by Banda *et al.*[8] The first two reactions show that the inputs S^1 and S^0 catalyze the transformation of the weight species into appropriate output species to represent multiplication.

4.6 Online learning of linear functions using DNAzymes

Lakin *et al.*[15] presented the design for a learning circuit similar to Banda *et al.*,[8] that can approximate real-valued linear functions of the form $f = w^{\mathrm{T}}x$ using DNAzymes (deoxyribozymes) (Fig. 3c). DNAzymes are a special class of DNA molecules that catalytically cleave specific nucleotide patterns in single-stranded RNA. A typical DNAzyme cleaving reaction contains a substrate S and a DNAzyme (Dz) where Dz cleaves S to

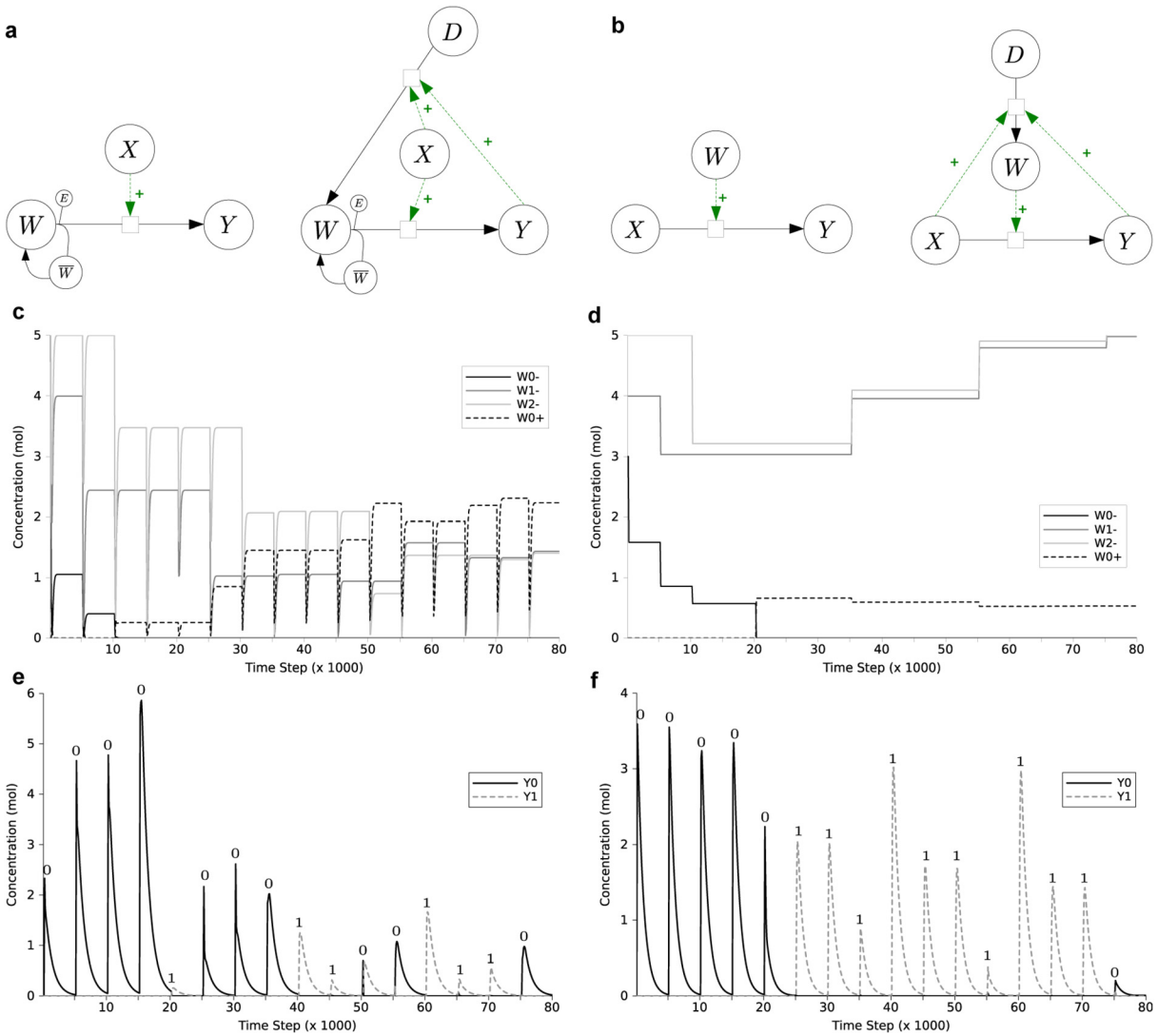

Fig. 6 (a) Reactions of the weight-loop perceptron. (b) Reactions of the weight-race perceptron. Both perceptrons are trained on the NAND function with weights beginning at $[W_0^{\ominus}] = 3$, $[W_1^{\ominus}] = 4$, $[W_2^{\ominus}] = 5$. (c) Training of the weight-loop perceptron. (d) Training of the weight-race perceptron. (e) Output of the weight-loop perceptron. (f) Output of the weight-race perceptron. This figure has been adapted from Banda *et al.*[8] with permission from MIT Press, copyright 2013.

form a product P ($S \xrightarrow{Dz} P + \text{waste}$).[57,58] The operation of their circuit could be seen as a combination of two distinct interacting sub-modules (although, in practice, both of them would be present in the same solution simultaneously): (i) the predictor subcircuit and (ii) the feedback subcircuit. These subcircuits can be seen as analogs to the forward and backpropagation stages of neural network learning. Both use a multiplier module implemented using DNAzymes as the basic computational element of their circuit.

They implemented the multiplier module as follows: first, the DNAzyme, initially in its inactive state (Dz.Iz), will be activated by the introduction of the activator species (A_{Dz}). This transforms the DNAzyme into its active state Dz. Dz has two distinct reaction pathways: (i) it can react with a substrate S_{Dz}.Z catalytically to release Z or (ii) it can react with a self-inhibitory substrate S_{Dz}.Iz, releasing Iz, converting itself back to its inactive state. The associated CRN is depicted in eqn (4).

$$\begin{aligned} A_{Dz} + Dz.Iz &\rightarrow Dz + A_{Dz}.Iz \\ Dz + S_{Dz}.Z &\xrightarrow{k} Dz + Z + \text{waste} \\ Dz + S_{Dz}.Iz &\xrightarrow{k} Dz + Iz \xrightarrow{\text{fast}} Dz.Iz \end{aligned} \tag{4}$$

A multiplier module implemented using DNAzymes.[15] The first reaction shows the activator species A_{Dz} flipping the state of the DNAzyme from the inactive (Dz.Iz) to the active state (Dz). The second and third reactions represent two different pathways for the DNAzyme. In the second reaction, the DNAzyme catalytically cleaves an input substrate molecule S_{Dz} into an output species Z. In the third reaction, the DNAzyme deactivates itself by cleaving a substrate that releases its deactivating species Iz.

At steady state, the concentrations of the corresponding species are related to each other as follows:

$$[Z]_{ss} = [A_{Dz}]_0 \frac{[S_{Dz}.Z]_0}{[S_{Dz}.Iz]_0}.$$

Considering the steady-state concentration of Z as the output, the above equation could be seen as a multiplication between the concentration of the activator species (input) and the concentration of the two substrate species (weight).

The predictor subcircuit carries out the forward step in the perceptron learning process by computing the function $f = \sum w_i x_i$ to obtain the output y. It uses the multiplier module described above to perform the multiplication operation. The feedback subcircuit updates the weights of the network analogous to the perceptron weight update algorithm. Since the weights in the circuit are represented by the ratio of the concentrations of the substrate species, they modify their concentrations accordingly to obtain the required weight change. This subcircuit is triggered by adding the target species $\hat{Z}$, which reacts annihilatively with the output species Z. The excess of either Z or $\hat{Z}$ triggers the production of the weight-changer species, which would then adjust the weights.

This work utilizes the cleaving properties of DNAzymes to implement an online learning algorithm in chemistry using reactions that are easily reproducible in a laboratory setting. Since each input goes through a disjoint reaction pathway, the network can easily accommodate additional inputs. However, because the weights and inputs were represented using concentrations of the chemical species, the circuit was not able to handle negative values, a drawback fixed in their later work discussed below. Further, as the weights were represented as a ratio of concentrations, the circuit elicited uncertain behavior when the value of weight was closer to zero.

4.7 Supervised learning using buffered DNA-strand displacement gates

Most of the drawbacks in Lakin *et al.*[15] were a by-product of the design choices involved in the representation of circuit variables. These issues were addressed in their follow-up work,[16] in which they use a purely DSD-based circuit forgoing DNAzymes. They introduce a new gate motif called the buffered DSD gate. In this setup, the gates are initially kept in an inactive form and could be activated conditionally upon the addition of an unbuffering strand. Further, the circuit could be cascaded by programming the upstream gate to contain the 'unbuffering strand' for a downstream gate. They utilized this motif to learn two-input linear functions of the form: $f(x_1, x_2) = w_1x_1 + w_2x_2$. The basic building block of this circuit is a multiplier module implemented as an amplifier circuit.

The implementation of the amplifier circuit involves two sets of reactions: (i) the input species X reacts with the activated gates to produce the output species Y; and (ii) input X is irreversibly consumed by a sink gate. Thus, amplification was achieved through competition with a gain factor equal to the ratio of the competing rates. Similar to their previous work, their network could also be divided into predictor and feedback subcircuits (Fig. 7a & b), both of which utilize the buffered amplifier as the building block. Further, to account for negative values, they represented the variables using the dual-rail format where values are modeled as a difference between two complementary species: $x = [X^+] - [X^-]$. The predictor subcircuit calculates the output function of the network $y = w_1x_1 + w_2x_2$ and the feedback subcircuit generates the gradient updates for the weights using the formula: $w_i = w_i + \alpha(d - y)x_i$. Notably, both the subcircuits require multiplication, addition, and subtraction operations which could be implemented using the amplifier circuit, and simple tricks such as output tying (different parts of the circuit producing the same output) and annihilation reactions. They further showed the construction of a three-way multiplier used by the feedback subcircuit (for calculating $\alpha(d - y)x_i$) using the buffered amplifier circuit.

A combination of VisualDSD[59] and MATLAB simulations were used to evaluate the DNA-based construction of the learning circuits. These results were compared to ODE simulations based on the mathematical process of weight updating specified in Fig. 7c. Simulations showed that the weight update trajectories of the DNA-based implementation and the corresponding ODE implementation matched closely, proving the validity of their network architecture.

4.8 A chemical feedforward neural network

Blount *et al.*[17] extended the work done by Banda *et al.*[14] on AASP to create the first-ever feedforward neural network in chemistry named feedforward chemical neural network (FCNN). The network has the same topology as the first classical neural network that simulates the XOR function with a two-node hidden layer and an output layer and can learn non-linearly separable binary functions such as XOR, XNOR, *etc*. The basic building blocks of the network are modified versions of AASP-style neurons embedded into cell-like compartments, which communicate by the permeation of the chemical species through the walls of the compartments.

Blount *et al.*[17] modified the design of the AASP model to introduce two new kinds of neurons: (i) the hidden chemical neuron (HCN) and (ii) the output chemical neuron (OCN) to support the feedforward and the backpropagation stages, respectively. FCNN represents its inputs and weights in a hybrid manner, *i.e.*, inputs being binary are represented by the existence (or absence) of the input species X_i, whereas the weights being real-valued are represented using the concentrations of the corresponding weight species W_i. The FCNN, similar to a neural network, operates in two phases, namely (i) the forward phase, where the inputs are propagated forward in the network through weight multiplication and integration to obtain the output, and (ii) the learning phase, where the error is calculated and the weights adjusted according to the backpropagation algorithm. The input-weight integration ($y = w_1x_1 + w_2x_2$) in the forward phase is achieved through competition between two reactions, where one reaction produces the output, and the other consumes it (eqn (5)). In the first reaction, the weight species W_i catalyzes the transformation of the input species X_i to

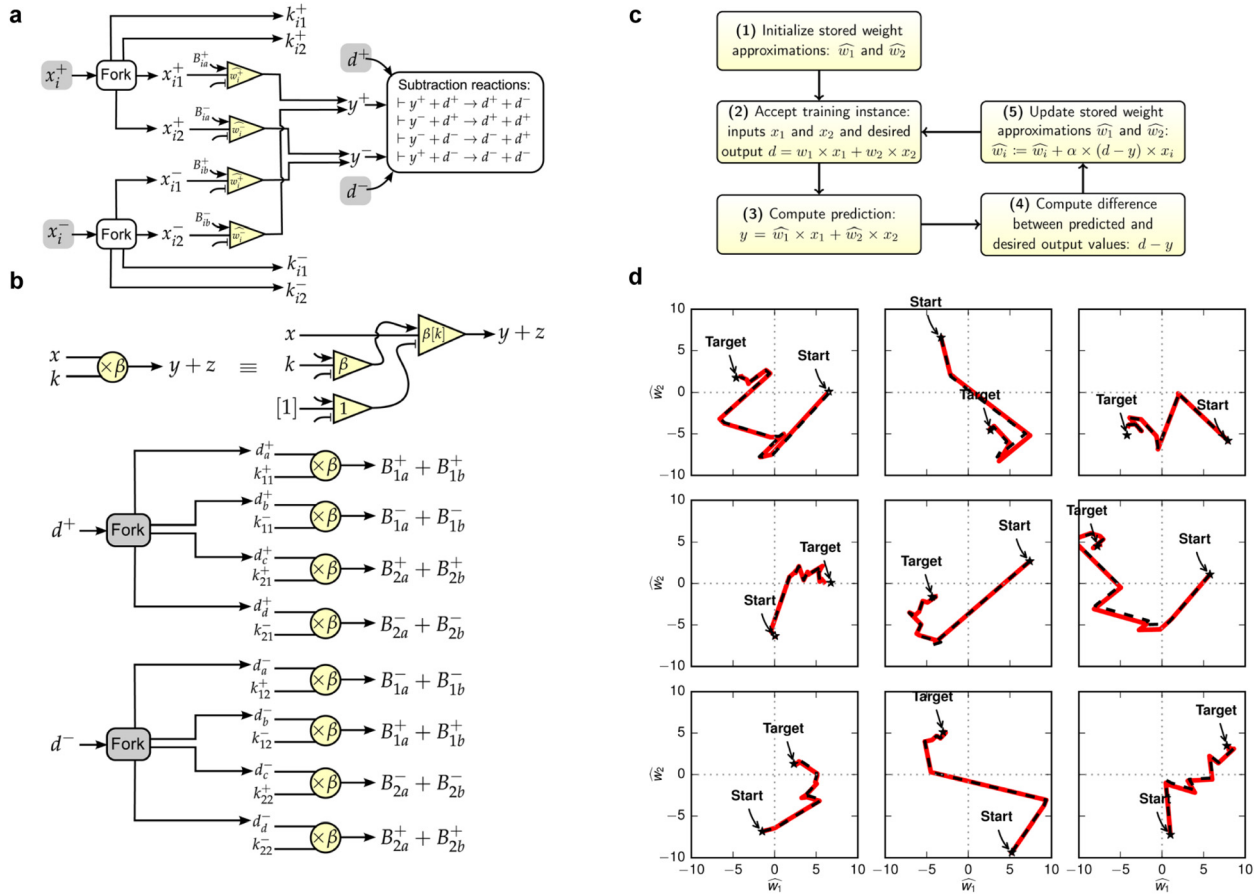

Fig. 7 Models and results of the "buffered DSD" DNA learning circuit architecture. (a) The predictor subcircuit. (b) The feedback subcircuit. (c) The operational scheme of the learning circuit. (d) Examples of training traces demonstrating learning in a DNA-based implementation. Red solid lines show the weight training results of the DNA learning circuit based on simulations of their ODEs, while the black dotted lines show the results of the operational scheme following the general process outlined in (c), which is without a DNA implementation. This figure has been reproduced from Lakin *et al.*[16] with permission from ACS, copyright 2016.

form the output Y and a feedback species X_iY. In the second reaction, the input X_i and the output Y annihilate with each other. The learning phase is triggered by the addition of target species $\hat{Y}$. First, the target and output species annihilate each other, simulating the error calculation. Then, the feedback species formed during the forward phase catalyzes the transformation of the remaining target/output (whichever is larger) species into the weight changer species $W^{\oplus}$ or $W^{\ominus}$ which in turn will adjust the neuron weights according to a variant of the perceptron learning algorithm (eqn (6)). Fig. 8 embellishes more details concerning the simulation arrangements and the two phases involved in the FCNN operation. The authors reported that their FCNN was able to learn the XOR function, reminiscent of the first classical implementation of learning the XOR function using backpropagation by Rumelhart.[60]

$$\begin{aligned} X_i &\xrightarrow{W_i} Y + X_iY \\ X_i + Y &\rightarrow \lambda \end{aligned} \tag{5}$$

CRN showing feedforward step of the FCNN network by Blount *et al.*[17] at the i^{th} node. In the first reaction, the weight species W_i catalyzes the transformation of the input species into the output species Y and a feedback species X_iY, which would be used during the backpropagation step. In the second reaction, the input species X_i and the output species Y combine with each other to form an unreactive waste (λ), otherwise known as a decay reaction. The combination of these two reactions simulates multiplication in the circuit through competition.

$$\begin{aligned} W^{\oplus} &\xrightarrow{X_iY} W_i \\ W^{\ominus} &\xrightarrow{X_iY} W_i^{\ominus} \\ W_i^{\ominus} + W_i &\rightarrow \lambda \end{aligned} \tag{6}$$

Weight adjustment step in Blount *et al.*[17] In the first reaction, the feedback species catalyzes the transformation of the positive weight adjuster species $W^{\oplus}$ into weight species W_i. In the second and third reactions, the negative weight changer species converts into a negative weight species for W_i, *i.e.*, $W_i^{\ominus}$, which then reacts annihilatively with the existing positive weight species W_i. Note that since we are dealing with binary functions, it isn't required to use negative weights.

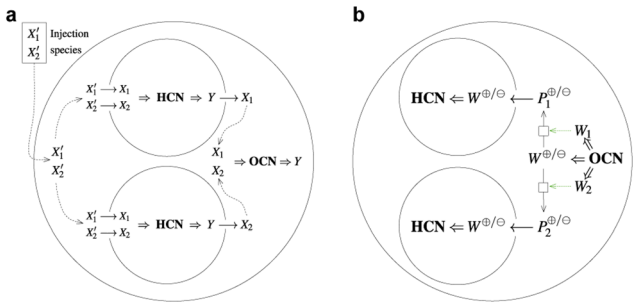

Fig. 8 Pictorial representation of the simulation chamber's arrangement and associated compartments. In both (a) and (b), we see a bigger outer compartment representing the OCN, while the two smaller compartments represent the two HCNs. (a) Depicts the forward phase of the FCNN described above. On the top-left, we see the input species being injected in their inactive forms into the OCN as represented by a X'_i (notice the superscript). They then enter the HCNs and are converted into their active form, and input-weight integration is performed. Outputs from the input-weight integration are then sent to the OCN as the input species. (b) It depicts the backpropagation stage, where the weight changer species are generated according to a modified version of the perceptron weight update rule. These weight changer species re-enter the HCN and adjust their weights appropriately according to the reactions shown in eqn (6). This figure has been reproduced from Blount *et al.*[17] with permission from MIT Press, copyright 2017.

4.9 Stochastic network simulation using stochastic CRNs

So far, the networks we encountered were simulated using deterministic CRNs whose dynamics follow the laws of mass action kinetics which require the reacting species to be present in large concentrations. However, microscopic environments such as the cell have a limited volume, restricting the use of deterministic kinetics. Poole *et al.*[10] uses stochastic CRNs for modeling learning in microscopic environments. In particular, they define two chemical versions of the Boltzmann machine (BM)[61] by utilizing the dynamics of stochastic CRNs to model its inference phase. Each node in a BM can exist in one of the two states ON or OFF. Reactions in the CBM are modeled as continuous time analogs of various sub-procedures of a BM implementation. For example, a single step of the Gibbs sampling procedure where a node is flipped (from 0 to 1) when all the neighboring nodes are 1 is modeled as a reaction where the species corresponding to the neighboring nodes of i^{th} node collectively catalyze the transformation of the species X_i^{OFF} to X_i^{ON}. To test if the implemented CBM is equivalent to its digital counterpart, they instantiated a CBM with weights trained *in silico* on the MNIST dataset[62] and demonstrated that CBM could generate images from a required distribution during inference.

In a similar vein, Singh *et al.*[19] proposed a stochastic CRN scheme for the Baum–Welch algorithm used to learn the parameters of a Hidden Markov model (HMM). It is a variant of the expectation–maximization (EM) algorithm, an iterative procedure that optimizes a given cost function $\mathcal{L}$ by first calculating the expected cost with the current parameters (expectation step) and then by updating the parameters to optimize the cost function until convergence. This work builds upon the reaction network schemes that can implement the EM algorithm that already exists in the literature, and we direct the reader to Virinchi *et al.*[18] for a detailed discussion.

4.10 Pattern recognition in the chemical medium through winner-take-all networks

Winner-take-all (WTA) refers to a reaction system where only one among all the input variables prevails at the steady state. Kim *et al.*[11] had previously proposed a WTA construction based on the competitive consumption of the enzyme by the involved reacting species.

Genot *et al.*[13] later elaborated upon this idea using both a DNA-only circuit as well as an enzymatic circuit. The authors focused on the principle that WTA relies on the competitive consumption of an amplifier and is a natural decision-making paradigm. In enzymatic circuits, this is often the polymerase, while in DNA-only circuits, this can be the fuel that operates an autocatalytic circuit. Their WTA network was demonstrated on the same 4-bit decision game as Qian *et al.*[12] using the VisualDSD simulator but additionally incorporated how the WTA paradigm could repair corrupted data. Notably, they remarked that the enzymatic variation of their circuit could run with just 16 strands, compared to the 23 strands required by a DNA-only implementation.

Cherry and Qian[5] demonstrated the design of a much larger DNA-only WTA network to perform pattern recognition in the chemical medium. The WTA network was first designed to recognize 3 × 3 patterns of English letters L and T (Fig. 9). Each cell in the pattern was represented using a unique binary chemical species, *i.e.*, the state of the cell is expressed by the presence or absence of a chemical species. The operation of this network could be divided into five stages (in what follows, the variable i represents the index of a cell in the pattern, and j represents the index of the pattern): (i) weight multiplication, $x_i w_{ij} (x_i \in \{0,1\}, \quad w_{ij} \in \mathcal{R}^+)$, where the input species X_i cata-

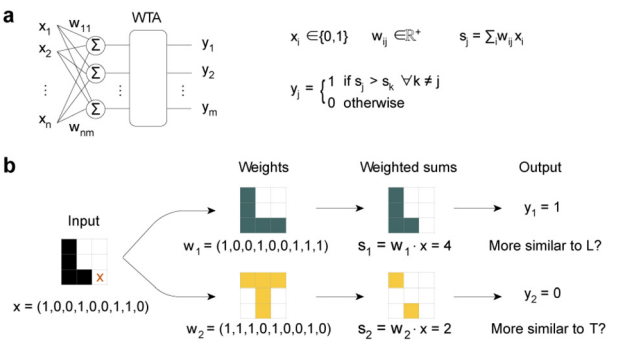

Fig. 9 Winner-take-all neural network for pattern recognition. (a) Operational diagram of the WTA circuit. The similarity with an expected pattern is evaluated. (b) Conceptual examples show the classification of two 3 × 3 patterns. Even if the circuit is given noisy information, it can still proceed by answering which input pattern is most similar. This figure has been reproduced from Cherry *et al.*[5] with permission from Springer Nature, copyright 2018.

lytically converted the weight species W_{ij} into the product species P_{ij}; (ii) summation, where all the product species pertaining to a single pattern were converted into a single weighted-sum species S_j; (iii) pairwise annihilation, where the S_j species reacted with each other annihilatively; (iv) signal restoration, which brought back the concentration of the remaining species to their original concentrations; (v) reporting, where several reporter reactions were used to convert each output to a fluorescent signal. The authors further demonstrated that their system is robust to noise by showing that it could handle tampered input patterns. Further, they extended this motif to identify handwritten digits from the MNIST[62] dataset of 10 × 10 patterns, and the resultant system was robust up to 30 (out of 100) bit flips in the input pattern. A follow-up paper from the same group described a loser-take-all network also.[63]

One of the advantages of a WTA-style network is that it doesn't require explicit training – which is often a cumbersome process, especially in chemistry – to perform classification as long as the explicit pattern to be recognized is known. The classification process in this motif is akin to comparing two images and tallying if they match, pixel by pixel. Such a comparison can be performed naturally in chemistry due to the parallelism inherent in chemical reactions. Therefore, one can see argue that WTA is a chemistry-friendly computational paradigm.

4.11 Binary-weight ReLU network

ReLU stands for rectified linear unit and is a popular activation function in the neural network literature. Vasic *et al.*[9] proposed a chemical system that implemented a three-layer network (an input layer, 1 hidden layer, and an output layer) called binary-weight ReLU, which uses ReLU as its activation function. It is a simplified multi-layer perceptron network with weights restricted to {1, −1}. The authors exploit the similarity between a class of coupled CRNs known as rate-independent CRNs and the ReLU function in the construction of a CRN for this network. Rate-independent CRNs, as the name suggests, are those where the steady-state concentrations of the reactant and product species are independent of the rate law and instead depend only on their stoichiometric coefficients. The reaction network shown below in eqn (7) depicts the implementation of a ReLU function using rate-independent CRNs.

$$\begin{aligned} X^+ &\rightarrow M + Y^+ \\ M + X^- &\rightarrow Y^- \end{aligned} \tag{7}$$

Implementation of the ReLU function using rate-independent reactions.

Their CRN (eqn (7)) represents the variables of the network in a dual-rail format, where the value of a variable is represented by the difference in the concentration between the two species preassigned for that variable. For example, the input value x_i of the network was represented by the species X^+ and X^- and its output value by Y^+ and Y^-. Fig. 10 shows the steady-state concentrations of the output species on a training problem from the IRIS dataset.

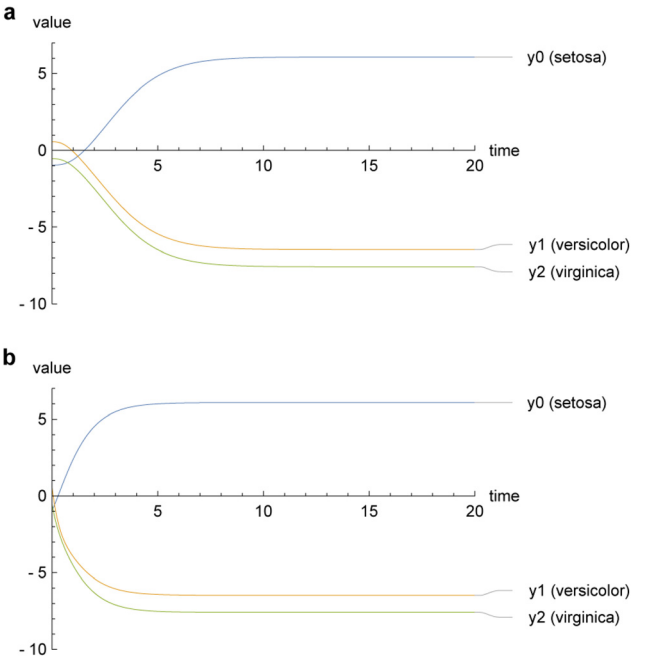

Fig. 10 Simulations of a CRN-based neural network implementing a ReLU activation function on the IRIS data. (a) Shows simulation results without simplification of unimolecular reactions. (b) Shows simulation results after initial unimolecular reactions of the ReLU implementation are compressed into downstream bimolecular reactions and concentrations. The results show a faster approach to steady-state and more accurate initial concentrations. This figure has been reproduced from Vasic *et al.*[9] with permission from ML Research Press, copyright 2020.

They showed that their proposed CRN implemented the binary-weight ReLU network by establishing the equivalence between a trained neural network and the ODE simulator of the compiled CRN with weights trained *in silico*. They further demonstrated this equivalence by testing the compiled CRN on various standardized datasets such as IRIS[64] and MNIST.[62]

4.12 Hebbian learning architectures using DNA

Fil *et al.*[22] used both CRNs and DNA-based constructions to describe a spiking neuron that closely modeled the activity of a biological neuron. Instead of a constant downstream signal, the spiking neuron produces spikes in signal output that attenuate at a set rate over time. The authors designed the CRN so that the output species of one neuron could be converted into the activation species of a different neuron. Using this model, they sought to demonstrate associative learning based on Hebbian theory through several examples. In colloquial terms, Hebbian theory is thought to describe the concept by which simultaneously firing neurons are more likely to wire together.

First, in a proof-of-concept demonstration, the authors defined a simple system of three neurons to demonstrate associative memory. The expected result was to observe the

influence of the weights of one connection upon the weights of another previously non-existent connection. Neurons A_1 and A_2 were inputs, and neuron B was an output. The activity of A_2 was initially independent of B (a spiking signal in A_2 did not trigger any activity in B), whereas a sufficiently high signal from A_1 did propagate the signal to B. Activation signals H are the edge weights, and a reversible reaction between an input A_n and the output B increases the concentration of each observed H when the signals of A_n and B are simultaneous. The example showed that through simultaneous firings of A_1 and A_2 together, the weight of A_2 and B increased, such that finally, when only A_2 and not A_1 fired, B was still triggered.

Next, the authors demonstrated the ability to discern up to five channels of firing neurons based on their signal frequency or how often a spiking signal was detected from inputs A_1 through A_5. Neurons that fired frequently would trigger an output more frequently, thus increasing the concentration of their respective H. Each channel was then ranked by its edge weights, and channels that fired frequently had heavier weights (higher H concentration) to the output, while channels that fired less frequently had lower weights (lower H concentration).

In addition and unique to this work, the theoretical constructions introduced memory decay and unsupervised learning, so relationships between neurons could be reprogrammed. This work also showed a DNA-based implementation for the presented CRNs *via* join-fork gates.[65] Join-fork gates describe a scalable DNA computation architecture from designing gates that can import multiple signals (join) and release multiple output signals (fork). The DNA-based constructions were simulated using VisualDSD[59] and demonstrated the same results for discerning frequency biases and temporal associations as those generated from simulating the CRNs.

4.13 DNA-based convolutional neural networks

Xiong *et al.*[6] implemented a convolutional neural network (CNN) *via* a DNA switching gate architecture to perform pattern recognition on 12 × 12 patterns. This is thus far the highest recorded input bandwidth of DNA-based neural networks. A CNN first convolves a local region (kernel), such as a 2 × 2 window, of data points into a feature. These features have the advantages that their input nodes share the same weight, feeding into the next layer of neural network nodes.

This work adopted the DNA switching gate architecture (Fig. 11a)[66] and used its secondary hairpin structure to modulate the strand displacement kinetics. The switching gate architecture is particularly adapted to this task as it contains a scalable design principle of modulating and repeating weights across different, distinct DNA-based gates implementing different nodes. On each side of the hairpin stem is a toehold domain and an output domain. To trigger the gate, an input strand has to attach to the toehold domain and then undergo a branch migration to displace the strand attached to the output domain (Fig. 3b). When the hairpin is formed, the toehold and output domain are also adjacent, thus allowing the branch migration to proceed. However, when a deactivating strand that could open the hairpin is present, the toehold and output domains are separated, and the branch migration becomes unfavorable (OFF state). An activating strand can be introduced, forming duplex waste with the deactivating strand to reform the hairpin and reinitiate the gate (ON state).

Through this motif, the authors demonstrated the construction of perceptron nodes required to design a DNA circuit implementing a CNN. The circuit was first trained *in silico*, and the evaluated weights were set by adjusting the hairpin stem of each DNA switching gate (Fig. 11b). Following the correct circuit setup, the authors demonstrated the classification of 32 distinct patterns. The patterns were further grouped into 4 subgroups (Arabic numerals, Chinese oracles, and English and Greek alphabets) of 8 patterns each. Each pattern was defined by the presence or lack of a DNA strand representing each of the 144 pixels of the pattern. A convolution of the pattern first determined the subgrouping of the pattern, while the second layer determined the precise pattern within that group (Fig. 11c & d). Due to the grouping, outputs could be identified as a multiplexed set of 4 × 8 fluorescent signals.

4.14 Towards a mathematical theory of CRN implementations of neural networks

So far, the works described above on molecular-scale learning have been particular constructions, *i.e.*, the methods and frameworks were tuned to the use case they were addressing. Anderson *et al.*[20] contended that, to fast-track the field's development, it is necessary to develop an appropriate mathematical framework that establishes the theoretical underpinnings of the CRNs that implement neural networks and learning, which could lead to a general theory of chemical learning. First, they prove that the ODE system associated with a class of CRNs can implement learning if and only if they manifest properties, such as the existence of unique fixed points and the possibility of faster convergence to those fixed points. Then, they prove the correctness of their theory by proving the equivalence between a CRN and a neural network that uses a "smoothed" ReLU activation function.

4.15 Nonlinear decision making with enzymatic neural networks

Datasets are often characterized by nonlinear dependencies between their inputs and outputs. CRN implementations that model such functions are often cumbersome and might require cascading existing reaction systems. Moreover, pure DNA reactions often suffer from leaky reactions and long reaction times, further exacerbated as the circuit grows. Enzyme-based implementations, on the other hand, are cleaner, less complicated, and enjoy faster reaction times. Okumura *et al.*[67] proposed the use of the PEN toolbox[48] (Fig. 3e) to design and implement an enzymatic neuron which adheres closely to the traditional perceptron model. The authors argued that the less leaky implementations of enzymatic circuits could simplify the composition of layers. Using this enzymatic neuron as a build-

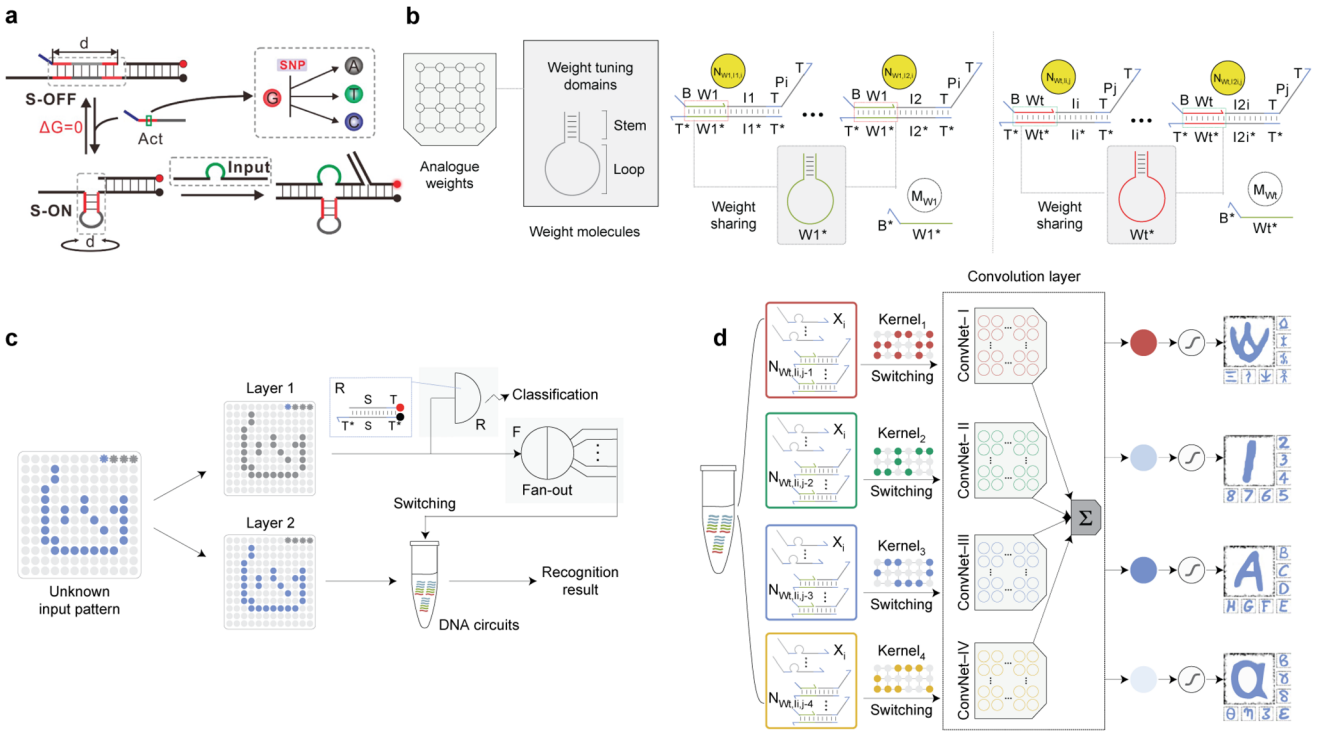

Fig. 11 The DNA switching gate architecture utilized to construct a molecular convolutional neural network. (a) The switching gate architecture has an energy landscape determined by its sequence and structure. Its primary advantage is the modulation of its reaction rate by simply adjusting the stem length of the hairpin structure. This promotes the scalability of its operation in large systems requiring multiple, distinct programmable species. This figure has been reproduced from Lai *et al.*[66] with permission from ACS, copyright 2018. (b) The advantages of the switching gate architecture are utilized to construct a convolutional neural network. Weight-sharing is achieved in the architecture by moderating the length of the hairpin stem to control the kinetics of binding the input. This compresses network complexity and implements the convolution step by weight-sharing within a convolved region of nodes. (c) The network is implemented to provide a two-step classification. First by category, then by the specific item within the category. (d) The DNA ConvNet was demonstrated to distinguish 32 different patterns divided as 8 patterns within each of 4 subgroupings. Each pattern was a 12 × 12 image represented by the presence or lack of a DNA strand representing each of the 144 pixels. This figure has been reproduced from Xiong *et al.*[6] with permission from Springer Nature, copyright 2022.

ing block, they constructed a multi-layer perceptron (MLP) with a two-node hidden layer, superposed by logic gates, to obtain nonlinear decision boundaries.

Using a space partitioning example, the authors demonstrated the circuit's capabilities to differentiate nonlinear boundaries. Their MLP aimed to find nonlinear decision boundaries according to the sum of two inputs. Of particular interest, this was finally performed using droplet microfluidics, which reduced working volumes by 10^5 and bolstered experimental throughput such that the circuit behavior could be characterized across 25 000 droplets. Each droplet contained some concentration of two inputs X_1 and X_2. The introduction of the MLP fluorescently labeled the sum of the inputs as one of three categories (α, β, or γ), thereby dividing the 2D concentration plane by three nonlinear decision boundaries.

4.16 Pattern recognition in the nucleation kinetics of non-equilibrium self-assembly

Evans *et al.*[21] studied the pattern recognition capabilities of the nucleation kinetics in a multifarious self-assembly process – a set of shared constituents forming many distinct structures. Their system comprised a mutually inert general-purpose tile set S and a set of interaction tiles named H, A, and M, the addition of which initiates the self-assembly process. Each of the interaction tiles H, A, and M necessitate the formation of H, A, and M patterns, respectively. The formation of the nucleation seed acts as the rate-limiting step in the self-assembly process, followed by a quick assembly around the seed to form the final pattern. Since the concentrations of the shared tiles drive the formation of the seed, those concentrations can be purported to be directly responsible for the kind of pattern formed, *i.e.*, the final pattern could be viewed as a complex function of the concentration vector $\mathcal{C}$ of the shared tiles. This work thus enables pattern recognition in this high-dimensional concentration space, where the concentration vector determines the nucleation seed and the pattern formed, leading to smooth decision regions in this high-dimensional concentration space. The high-dimensionality aspect, however, comes naturally as a part of the system design and should not be treated as a feature, *i.e.*, systems with lower dimensionality will not have simpler implementations.

5. Outlook and drawbacks

As the field is very much in its nascence, most works focused on building architectures "that just work" for the use case under consideration rather than developing general-purpose architectures and algorithms. Further, problems such as leaks, slow reaction times, and reaction cross-talk prevent the implementations from simulating even moderate-sized neural networks.

In this work, we tried to put in context these fragmented adaptations of machine learning at the molecular scale. In doing so, we bring to light several commonalities in seemingly disparate attempts at modeling said *learning*. For example, most works consider representing the values of weights and inputs separately using the concentrations of real variables. Many works produce appropriate weight-changer species to readjust their weights. Despite this, no single, comprehensive experiment stands out as direct evidence for "learning" (*e.g.*, ANN learning) at the molecular scale. The performance advantages of ANNs have been due to the sheer scale of their networks and the data bandwidth available to train them. Despite the intriguing prospects of parallelism and biocompatibility of DNA computing, these advantages are not present within the existing molecular computing technologies. Although, it is also not fair or realistic to expect a simple repetition of machine learning in a different medium. It is, however, undeniable that the questions of scalability in molecular-scale learning systems predetermine its practical usage.

A scalable molecular computing architecture for implementing neural network-like computation would desire the availability of several magnitudes more distinct DNA strands operating within the same volume than what has been demonstrated. There are two closely related challenges to combat in the scaling up: (i) orthogonality and (ii) leak. Orthogonality is the property that the designed DNA sequences that follow two disparate reaction pathways remain mutually inert to each other in a well-mixed solution. This is necessary to design reactions with negligible cross-talk and ensure the system works as intended. While the software packages such as NUPACK[68] and the more recent SeqWalk[69] help design large sets of orthogonal species, there is a need to develop novel computational tools for various DNA computing architectures, such as the seesaw compiler[70] that discover not only orthogonal species but also orthogonal reaction pathways at the scale necessary to build neural networks of substantial depth and complexity.

Reaction leak is another closely related issue. If the species are not sufficiently orthogonal, spurious reactions can happen, which can cause the system to diverge. These can also frequently occur due to synthesis errors, and PAGE purification of strands is a baseline prerequisite for any DNA computing circuit. Yet even when all the design principles are correctly accounted for, leaks can still be a prominent contributor to errors due to a phenomenon called fraying, where the last few bases at either end of a DNA duplex bind poorly and can transiently expose an unintended toehold resulting in a spurious strand displacement. Such strands could begin propagating errors downstream in the circuit, making it unviable. One of the simpler modifications is to design domains at a sequence level such that all the duplexes are terminated by G–C bonds, which are stronger than the A–T bonds. "Clamping"[71] is another frequently used technique to mitigate leakage from fraying where each duplex is extended by 1 to 3 bases, so the fraying happens at an unimportant domain instead. This technique was applied in the examples of neural network-like circuits discussed in this review.[5,6,12]

Other strategies to combat leak include more drastic design changes, such as running a shadow circuit ("shadow cancellation"),[72] or fundamental changes to the architecture like lengthening domains,[71,73] catalytic hairpin assembly (CHA),[74,75] or the "junction substrate".[76] Compared to clamping, which is straightforwardly applicable, implementations of new architectures into existing examples can be more complicated. While architectures are developed to be computationally universal, it is not guaranteed that there can be a straightforward implementation of the critical principles of neural networks, like weight and thresholding, which happened to have simple implementations in seesaw gates and switching gates. Such questions remain to be objectively answered. Strategies like shadow cancellation or CHA involve making principled decisions on positioning domains based on predicted leak sites. The principles applied in these strategies to counteract leaks could also limit the sequence variability and introduce design overheads that could hamper neural network implementations that are already vying and hard-pressed for room to scale upwards. It would be interesting to see, rather than a race for scale, the variations of architectures in implementing only a single perceptron unit.

Another common theme regarding the DNA implementations of learning is the reliance on training the circuits *in silico* and then translating the resultant weights into the concentrations of the chemical species. In other words, these networks are static instances of a trained neural network. They should be capable of autonomous learning to build networks similar to biochemical machinery. It should be noted that such a discussion should include caution, as looking so far ahead could be counterproductive to grounding the ideas on what is possible and what can be achieved. For instance, weights were often designed as the concentrations within seesaw gates, whereas they are integrated into the strand design in the switching gates architecture. While the former appears more compatible with the goal of adaptation, the latter demonstrated the largest-scale instance of neural network-like computing in molecular computing thus far.

Fundamentally, being able to re-train a DNA-based circuit reinvokes the process of implementing the loss function from the perceptron training algorithm. While initial runs of a DNA circuit proceed serially, the circuit must also be expected to run iteratively, repeating the same weight update process after each training input. The PEN toolbox has previously demonstrated such temporal, oscillatory behavior by demonstrating predator–prey dynamics, for example, as CRNs.[48] While this

behavior is periodic, it offers some promise in the context of DNA-based neural networks – the concentration of one species, such as a weight species, can be modulated based on observed concentrations of another species, such as the loss value. More recently, Lapteva, Sarraf, & Qian[77] also demonstrated temporal DNA logic circuits using only DNA-based reactions, providing an architecture for comparing the time of arrival between two signals leading to strategies that define the order of learning events in DNA circuits. The surveyed work by Lakin *et al.*[15] proposed an amplifier circuit that can be dynamically modified, but it has not been demonstrated experimentally yet. In a broad sense, designing a DNA circuit that can run iteratively remains a challenging problem.

While grounding our expectations for the future in the field, researchers should imbue an affinity towards scalable general-purpose architectures that are robust to noise. Technologies other than pure DNA-based circuits, such as enzymes from the PEN toolbox, strand-displacing polymerase,[78] microfluidics,[79] or reaction–diffusion systems[80] could introduce dynamics that may be otherwise unwieldy and potentially impossible to implement using DNA strands alone. To revisit our previous speculation, adding nicking and ligation enzymes could allow the switching gate architecture to change its weight-determining hairpin. Reaction–diffusion systems can be naturally oscillatory, whereas microfluidics has stepwise automation, filtration, and temperature control mechanisms that could stabilize DNA-based reactions and outsource the circuit complexity.

While instances of applications are still few, Lopez *et al.* demonstrated a potential application of DNA-based neural network-like computation for early cancer diagnostics – an indication of the potentially transformative impacts of this discipline. Correlation data of gene expression profiles to cancer diagnoses were used as training data to determine the weights of a linear classifier called the support vector machine (SVM), a construction similar to a single perceptron. *In silico* training found minimum representations of each genetic profile. When RNA inputs were input into the system as indicators for each gene, the trained SVM could determine a weighted likelihood for cancer by comparing the relative presence of each gene, affected by the trained weights. This construction similarly classified infections as caused by viral or bacterial pathogens.

Overall, we view molecular-scale, DNA-based neural networks as integrating some of the furthest state-of-the-art techniques in DNA computing. Furthermore, we believe a large, unexplored portion of DNA computing techniques waits to be evaluated for their applicability to program neural networks. There has been a recent growth in the publication of theoretical models of CRNs that implement neural network computation, and attempts at implementing, which has traditionally been the trend in DNA computing, should soon follow. Fundamental techniques in DNA computing that advance the scale and robustness of its circuits continue to be discovered, and we expect this to accelerate the developments in molecular learning.

6 Conclusion

In the last two decades, the field of molecular computing has made many outstanding achievements, both theoretical as well as in laboratory demonstrations. But what should be the future goals and aspirations of this emerging field?

Through this review, we argue that one of the noteworthy goals of the molecular computing community should be to achieve programmability in chemical systems through learning and adaptation. To estimate where we stand currently, we discussed several works from the DNA computing community that make significant strides towards achieving this goal through simulation or practical implementations. We further used these works to understand the advantages and shortcomings of their varied techniques. These applications described their potential for recognition tasks, often by encrypting digital image data into a molecular form. However, it should not be expected that computational processing using chemical mediums should hope to triumph against silicon computation. Applications that are truly biologically relevant and more naturally suited to the medium, such as comprehensive biosensing diagnostic technologies or intelligent control systems for drug delivery agents that automate diagnostic and therapeutic cycles of medical care, are more realistic expectations of the field that may await progress far beyond the horizon.

Author contributions

R. N., D. F., and J. R. conceptualized the article. R. N. and D. F. investigated previous works for the manuscript and also wrote and edited the manuscript with expert guidance, suggested edits, and supervision from J. R.

Conflicts of interest

There are no conflicts to declare.

Acknowledgements

The authors would like to acknowledge funding support from the National Science Foundation under grants no. 1909848 and 2113941 to J. R.

References

1 D. Bray, *Nature*, 1995, **376**, 307–312.
2 T. M. Hennessey, W. B. Rucker and C. G. McDiarmid, *Anim. Learn. Behav.*, 1979, **7**, 417–423.
3 C. T. Fernando, A. M. L. Liekens, L. E. H. Bingle, C. Beck, T. Lenser, D. J. Stekel and J. E. Rowe, *J. R. Soc., Interface*, 2009, **6**, 463–469.

4 A. Hjelmfelt, E. D. Weinberger and J. Ross, *Proc. Natl. Acad. Sci. U. S. A.*, 1991, **88**, 10983–10987.
5 K. M. Cherry and L. Qian, *Nature*, 2018, **559**, 370–376.
6 X. Xiong, T. Zhu, Y. Zhu, M. Cao, J. Xiao, L. Li, F. Wang, C. Fan and H. Pei, *Nat. Mach. Intell.*, 2022, **4**, 625–635.
7 R. Lopez, R. Wang and G. Seelig, *Nat. Chem.*, 2018, **10**, 746–754.
8 P. Banda, C. Teuscher and M. R. Lakin, *Artif. Life*, 2013, **19**, 195–219.
9 M. Vasic, C. Chalk, S. Khurshid and D. Soloveichik, *Proceedings of the 37th International Conference on Machine Learning*, MLResearchPress, 2020, pp. 9701–9711.
10 W. Poole, A. Ortiz-Munoz, A. Behera, N. S. Jones, T. E. Ouldridge, E. Winfree and M. Gopalkrishnan, *DNA Computing and Molecular Programming*, Springer, 2017, pp. 210–231.
11 J. Kim, J. Hopfield and E. Winfree, *Advances in Neural Information Processing Systems*, MIT Press, 2004, vol. 17.
12 L. Qian, E. Winfree and J. Bruck, *Nature*, 2011, **475**, 368–372.
13 A. J. Genot, T. Fujii and Y. Rondelez, *J. R. Soc., Interface*, 2013, **10**, 20130212.
14 P. Banda and C. Teuscher, *International Conference on Unconventional Computation and Natural Computation*, Springer, 2014, pp. 14–26.
15 M. R. Lakin, A. Minnich, T. Lane and D. Stefanovic, *J. R. Soc., Interface*, 2014, **11**, 20140902.
16 M. R. Lakin and D. Stefanovic, *ACS Synth. Biol.*, 2016, **5**, 885–897.
17 D. Blount, P. Banda, C. Teuscher and D. Stefanovic, *Artif. Life*, 2017, **23**, 295–317.
18 V. V. Muppirala, A. Behera and M. Gopalkrishnan, *DNA Computing and Molecular Programming*, Springer, 2018, pp. 189–207.
19 A. Singh, C. Wiuf, A. Behera and M. Gopalkrishnan, *DNA Computing and Molecular Programming*, Springer, 2019, pp. 54–79.
20 D. F. Anderson, B. Joshi and A. Deshpande, *J. R. Soc., Interface*, 2021, **18**, 20210031.
21 C. G. Evans, J. O'Brien, E. Winfree and A. Murugan, 2022, preprint, arXiv: 2207.06399 [cond-mat.dis-nn], DOI: **10.1021/acssynbio.1c00625**.
22 J. Fil, N. Dalchau and D. Chu, *ACS Synth. Biol.*, 2022, **11**(6), 2055–2069.
23 D. Fan, J. Wang, E. Wang and S. Dong, *Adv. Sci.*, 2020, **7**, 2001766.
24 Y. Hua, J. Ma, D. Li and R. Wang, *Biosensors*, 2022, **12**, 183.
25 Y.-J. Chen, B. Groves, R. A. Muscat and G. Seelig, *Nat. Nanotechnol.*, 2015, **10**, 748–760.
26 Q. Hu, H. Li, L. Wang, H. Gu and C. Fan, *Chem. Rev.*, 2018, **119**, 6459–6506.
27 E. Darley, J. K. D. Singh, N. A. Surace, S. F. Wickham and M. A. Baker, *Genes*, 2019, **10**, 1001.
28 N. Stephanopoulos, *Chem*, 2020, **6**, 364–405.
29 L. Shen, P. Wang and Y. Ke, *Adv. Healthcare Mater.*, 2021, **10**, 2002205.
30 C. Zhang, Y. Zhao, X. Xu, R. Xu, H. Li, X. Teng, Y. Du, Y. Miao, H.-c. Lin and D. Han, *Nat. Nanotechnol.*, 2020, **15**, 709–715.
31 A. R. Chandrasekaran, J. A. Punnoose, L. Zhou, P. Dey, B. K. Dey and K. Halvorsen, *Nucleic Acids Res.*, 2019, **47**, 10489–10505.
32 O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed and H. Arshad, *Heliyon*, 2018, **4**, e00938.
33 A. Krizhevsky, I. Sutskever and G. E. Hinton, *Commun. ACM*, 2017, **60**, 84–90.
34 D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel and D. Hassabis, *Nature*, 2016, **529**, 484–489.
35 J. Ker, L. Wang, J. Rao and T. Lim, *IEEE Access*, 2017, **6**, 9375–9389.
36 M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald and E. Muharemagic, *J. Big Data*, 2015, **2**, 1–21.
37 L. Deng, D. Yu, *et al.*, *Found. Trends Signal Process.*, 2014, **7**, 197–387.
38 S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen and S. S. Iyengar, *ACM Comput. Surv.*, 2018, **51**, 1–36.
39 T. Lin, Y. Wang, X. Liu and X. Qiu, *AI Open*, 2022, **3**, 111–132.
40 A. Esteva, K. Chou, S. Yeung, N. Naik, A. Madani, A. Mottaghi, Y. Liu, E. Topol, J. Dean and R. Socher, *NPJ Digit. Med.*, 2021, **4**, 5.
41 X. Su, S. Xue, F. Liu, J. Wu, J. Yang, C. Zhou, W. Hu, C. Paris, S. Nepal, D. Jin, *et al.*, *IEEE Trans. Neural Netw. Learn. Syst.*, 2022, 1–21.
42 K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, *IEEE Signal Process. Mag.*, 2017, **34**, 26–38.
43 J. Schmidhuber, *Neural Netw.*, 2015, **61**, 85–117.
44 D. E. Rumelhart, R. Durbin, R. Golden and Y. Chauvin, *Backpropagation: Theory, Architectures and Applications*, Lawrence Erlbaum Hillsdale, NJ, USA, 1995, pp. 1–34.
45 W. S. McCulloch and W. Pitts, *Bull. Math. Biophys.*, 1943, **5**, 115–133.
46 F. Rosenblatt, *Psychol. Rev.*, 1958, **65**, 386.
47 L. Ma and J. Liu, *iScience*, 2020, **23**, 100815.
48 K. Montagne, R. Plasson, Y. Sakai, T. Fujii and Y. Rondelez, *Mol. Syst. Biol.*, 2011, **7**, 466.
49 D. Soloveichik, G. Seelig and E. Winfree, *Proc. Natl. Acad. Sci. U. S. A.*, 2010, **107**, 5393–5398.
50 D. Y. Zhang and G. Seelig, *Nat. Chem.*, 2011, **3**, 103–113.
51 F. C. Simmel, B. Yurke and H. R. Singh, *Chem. Rev.*, 2019, **119**, 6326–6369.
52 F. Fages, S. Gay and S. Soliman, *Theor. Comput. Sci.*, 2015, **599**, 64–78.
53 J. J. Hopfield, *Proc. Natl. Acad. Sci. U. S. A.*, 1982, **79**, 2554–2558.

54 L. Qian and E. Winfree, *J. R. Soc., Interface*, 2011, **8**, 1281–1297.
55 L. Qian and E. Winfree, *Science*, 2011, **332**, 1196–1201.
56 P. Dittrich, J. Ziegler and W. Banzhaf, *Artif. Life*, 2001, **7**, 225–275.
57 J. C. Achenbach, W. Chiuman, R. P. G. Cruz and Y. Li, *Curr. Pharm. Biotechnol.*, 2004, **5**, 321–336.
58 L. Gong, Z. Zhao, Y.-F. Lv, S.-Y. Huan, T. Fu, X.-B. Zhang, G.-L. Shen and R.-Q. Yu, *Chem. Commun.*, 2015, **51**, 979–995.
59 M. R. Lakin, S. Youssef, F. Polo, S. Emmott and A. Phillips, *Bioinformatics*, 2011, **27**, 3211–3213.
60 D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Nature*, 1986, **323**, 533–536.
61 G. Hinton, *Encyclopedia of Machine Learning and Data Mining*, Springer US, Boston, MA, 2014, pp. 1–7.
62 Y. LeCun, *The MNIST Database of Handwritten Digits*, 1998, **https://yann.lecun.com/exdb/mnist/**.
63 K. R. Rodriguez, N. Sarraf and L. Qian, *ACS Synth. Biol.*, 2021, **10**, 2878–2885.
64 D. Dua and C. Graff, *UCI Machine Learning Repository*, 2017, **https://archive.ics.uci.edu/ml**.
65 L. Cardelli, *Math. Struct. Comput. Sci.*, 2013, **23**, 247–271.
66 W. Lai, L. Ren, Q. Tang, X. Qu, J. Li, L. Wang, L. Li, C. Fan and H. Pei, *ACS Nano*, 2018, **12**, 7093–7099.
67 S. Okumura, G. Gines, N. Lobato-Dauzier, A. Baccouche, R. Deteix, T. Fujii, Y. Rondelez and A. J. Genot, *Nature*, 2022, **610**, 496–501.
68 J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks and N. A. Pierce, *J. Comput. Chem.*, 2011, **32**, 170–173.
69 G. Gowri, K. Sheng and P. Yin, 2022, bioRxiv:2022.07.11.499592, DOI: **10.1101/2022.07.11.499592**.
70 A. J. Thubagere, C. Thachuk, J. Berleant, R. F. Johnson, D. A. Ardelean, K. M. Cherry and L. Qian, *Nat. Commun.*, 2017, **8**, 14373.
71 B. Wang, C. Thachuk, A. D. Ellington, E. Winfree and D. Soloveichik, *Proc. Natl. Acad. Sci. U. S. A.*, 2018, **115**, E12182–E12191.
72 T. Song, N. Gopalkrishnan, A. Eshra, S. Garg, R. Mokhtar, H. Bui, H. Chandran and J. Reif, *ACS Nano*, 2018, **12**, 11689–11697.
73 C. Thachuk, E. Winfree and D. Soloveichik, *International Workshop on DNA-Based Computers*, Springer, 2015, pp. 133–153.
74 X. Chen, N. Briggs, J. R. McLain and A. D. Ellington, *Proc. Natl. Acad. Sci. U. S. A.*, 2013, **110**, 5386–5391.
75 Y. S. Jiang, S. Bhadra, B. Li and A. D. Ellington, *Angew. Chem.*, 2014, **126**, 1876–1879.
76 X. Sun, B. Wei, Y. Guo, S. Xiao, X. Li, D. Yao, X. Yin, S. Liu and H. Liang, *J. Am. Chem. Soc.*, 2018, **140**, 9979–9985.
77 A. P. Lapteva, N. Sarraf and L. Qian, *J. Am. Chem. Soc.*, 2022, **144**, 12443–12449.
78 S. Shah, J. Wee, T. Song, L. Ceze, K. Strauss, Y.-J. Chen and J. Reif, *J. Am. Chem. Soc.*, 2020, **142**, 9587–9593.
79 W. Lee, M. Yu, D. Lim, T. Kang and Y. Song, *ACS Nano*, 2021, **15**, 11644–11654.
80 N. Dalchau, G. Seelig and A. Phillips, *International Workshop on DNA-Based Computers*, Springer, 2014, pp. 84–99.
81 L. M. Adleman, *Science*, 1994, **266**, 1021–1024.