Supervised learning in an adaptive DNA strand displacement circuit

Matthew R. Lakin[1,2] and Darko Stefanovic[1,2]

[1] Department of Computer Science, University of New Mexico, NM 87131, USA
[2] Center for Biomedical Engineering, University of New Mexico, NM 87131, USA

{mlakin,darko}@cs.unm.edu

Abstract. The development of DNA circuits capable of adaptive behavior is a key goal in DNA computing, as such systems would have potential applications in long-term monitoring and control of biological and chemical systems. In this paper, we present a framework for adaptive DNA circuits using buffered strand displacement gates, and demonstrate that this framework can implement supervised learning of linear functions. This work highlights the potential of buffered strand displacement as a powerful architecture for implementing adaptive molecular systems.

1 Introduction

Implementing adaptive behaviors, such as supervised learning, is a key challenge for the fields of molecular computing and synthetic biology. Addressing this challenge would enable the development of molecular computing solutions to important practical applications, such as the detection of emerging pathogens [1] whose signatures mutate over time. Furthermore, the development of molecular computing systems that are capable of operating over an extended period of time would advance the state of the art of molecular circuit design, as most current systems are single-use devices.

Previous experimental work has shown that neural networks may be implemented using DNA strand displacement circuits [2] comprising "seesaw" gates [3, 4]. However, in that work the neural network was trained *in silico* and each instance of the experimental system could only be used one time. Our prior theoretical work has demonstrated that a biochemical system assuming hypothetical, DNAzyme-like reactions can learn a class of linear functions [5], and other work has shown that high-level artificial chemistries can learn to implement Boolean functions [6] and perceptron-like classification tasks [7]. Here we present a design for an adaptive, reusable DNA learning circuit based on the framework of four-domain DNA strand displacement reactions, which has been shown to be capable of implementing arbitrary chemical reaction networks in a concrete biochemical system [8]. Thus, this paper offers a route to an experimental realization of adaptive DNA computing systems.

The remainder of this paper is organized as follows. In Section 2, we introduce buffered DNA strand displacement systems and illustrate their use for implementing adaptive systems. In Section 3, we present an adaptive buffered amplifier, which is a key component of the learning circuit that we present in Section 4. We present results

from computational simulations of this learning circuit in Section 5, and conclude with a discussion in Section 6.

2 Buffered DNA strand displacement for adaptive systems

The idea of buffered DNA strand displacement gates was introduced (as "curried" gates) by Cardelli [9]. This idea was further developed in the context of implementing DNA oscillators with robust long-term kinetics [10]. In this section we recap the principle of buffered strand displacement gates and show how they can be used to implement adaptive molecular systems.

The basic principle of buffered strand displacement is illustrated in Figure 1. In this paper we base our gate designs on the "four-domain" encoding of abstract chemical reaction networks into DNA strand displacement, developed by Soloveichik [8]. In the four-domain encoding, each abstract species X is represented by three domains $Xa^$, Xb and $Xc^$. We write "?" to denote a "history domain" whose identity is irrelevant for the operation of the gate (however, the history domains for the product species must be freshly generated for each gate, to avoid crosstalk). The key difference is that, rather than initializing the system with populations of active gates capable of accepting input strands directly, a buffered system is initialized with a *buffer* of inactive gates that cannot initially accept input strands. A population of "unbuffering" strands must first be introduced, which (irreversibly) activate a subset of the buffered gates, so that they can accept input strands as normal. We design the gates so that an additional copy of each gate's unbuffering strand is released along with the gate's outputs, to maintain a (roughly) constant population of active gates. Furthermore, since the gates in the buffer are inactive, the population of inactive buffered gates can be replenished, either at intervals or continuously, without significantly affecting the reaction kinetics. If waste products were removed, to prevent them from accumulating, this could allow buffered strand displacement systems to run indefinitely.

We will write a buffered strand displacement gate that implements the reaction $\overline{R} \rightarrow \overline{P}$ with buffer species B using the notation $B \vdash \overline{R} \rightarrow \overline{P}$, and we will use the $\varnothing$ symbol to denote an empty (multi)set of reactants or products.

We propose to use buffered strand displacement gates to implement adaptive systems. To this end, we note that altering the concentration of unbuffering strands that are available for a given buffered gate provides a means of controlling the rate of the gate's reaction: if more copies of a given reaction gate are activated from the buffer, it will emulate a faster reaction. (This is a simpler approach than engineering toehold binding energies or developing remote toehold systems [11].) Furthermore, in addition to regenerating its own unbuffering strand, a given buffered gate may also release unbuffering strands for other gate populations as part of its output. This crucial fact enables one buffered gate to adjust the number of active gates of a second kind, thereby controlling the rate of the second buffered gate's reaction.

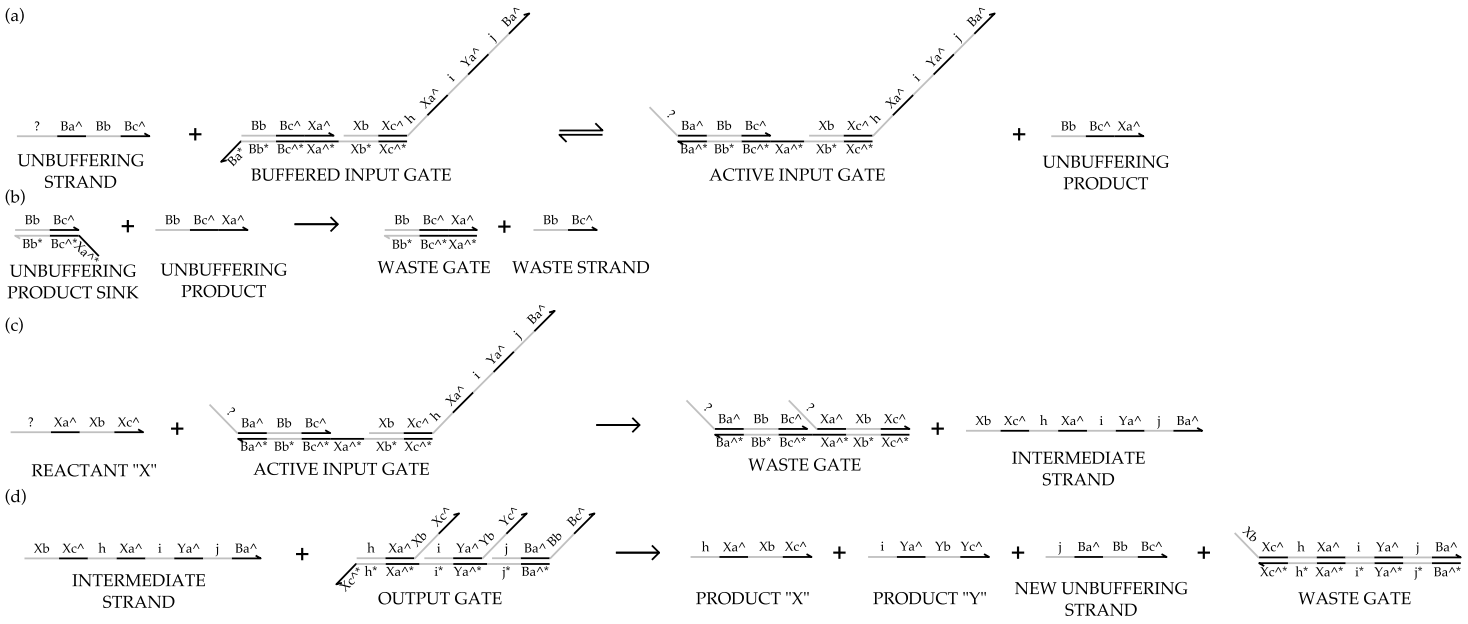

Fig. 1. Example of a buffered four-domain DNA strand displacement gate, implementing the buffered reaction $B \vdash X \rightarrow X + Y$. **a.** The first reaction is the unbuffering reaction, in which an unbuffering strand binds to a buffered gate and reversibly produces an active input gate. The binding toehold for the X reactant strand is exposed in the active input gate. **b.** To make the unbuffering reaction effectively irreversible, an additional sink gate for the strand released by the unbuffering reaction is provided (this gate is not required in standard four-domain chemical reaction network encodings). **c.** A reactant strand X can bind to an active input gate via the $Xa^{\wedge}$ toehold, irreversibly displacing an intermediate strand. **d.** The released intermediate strand binds to the output gate and releases all of the products in a single strand displacement reaction. Here, in addition to the product strands X and Y, an additional output, B, is generated, which is a new copy of the unbuffering strand for this gate. Thus, a new gate will be activated from the buffer to replace the gate that was consumed to execute this reaction.

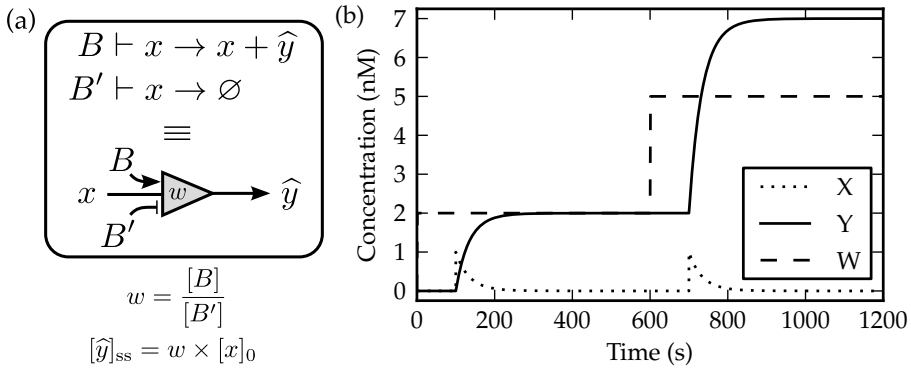

Fig. 2. Buffered amplifier design and ODE simulation data. **a.** The buffered amplifier consists of two buffered reactions: a catalytic reaction that generates the output and a degradation reaction that removes the input from the system. The ratio of the concentrations of active gates from the two buffers corresponds to the gain of the amplifier. **b.** Data from an ODE simulation of the buffered amplifier with multiple sequential input additions and dynamic control of amplifier gain. The initial concentration of B' was 100 nM and the initial concentration of B was 200 nM, which sets the gain, W, to be 2.0. Addition of the X input species (1 nM) at $t = 100$ s causes the amplifier to generate the output species Y at a concentration of 2 nM, as expected. At $t = 600$ s, an additional 300 nM of the unbuffering strand B was added, increasing the amplifier's gain (W) by 3.0. Then, subsequent addition of X (1 nM) at $t = 700$ s produced a further 5 nM of the output species Y, showing that the gain had indeed been increased to 5.0, as expected.

3 An adaptive, buffered amplifier

In this section, we will illustrate the use of buffered strand displacement gates to implement an adaptive amplifier whose gain can be dynamically adjusted. This contrasts with previous work on DNA strand displacement-based amplifiers [12, 3, 13, 4, 14], which relied on hard-coding the gain of the system in the initial species concentrations, and gave no consideration to reusability or autonomously adjusting the gain of the amplifier. In addition to providing an example of a buffered strand displacement system whose result can be controlled by adjusting the provision of unbuffering strands, this circuit design motif will be a key component of our learning circuit design.

Our design for a reusable, adaptive, buffered strand displacement-based amplifier consists of two buffered strand displacement gates:

$$B \vdash x \to x + y \qquad\qquad B' \vdash x \to \varnothing.$$

The first gate is a "catalyst" gate that uses the input strand x to catalyze release of the output strand y, and the second gate is a "degradation" gate that removes the input strand x from the system by consuming it without releasing any output species (except for a new activating strand B'). This design draws on ideas introduced by Zhang and Seelig [14]: the ratio between the effective rates of the first and second gates controls the number of output molecules y that each input molecule x can produce before it is

irreversibly consumed, and therefore this ratio controls the gain of the amplifier. Thus, in the amplifier circuit, the concentration ratio $\frac{[B]}{[B']}$ controls the gain of the amplifier. More specifically, if we write $[z]_0$ for the initial concentration of species z and $[z]_{ss}$ for the steady state concentration of z (assuming that a steady state exists), then we would expect that $[y]_{ss} = \frac{[B]_0}{[B']_0} \times [x]_0$. A graphical shorthand for this circuit design element is presented in Figure 2(a). Our approach to implementing the amplifier circuit motif is also related to the "ideal gain blocks" developed by Oishi and Klavins [15], the main difference being that our approach produces a quiescent final state, whereas their approach produces a steady state in which production and degradation of the output species balance. Thus, our approach prevents the supply of input species being constantly drained as in [15]. However, our approach does require some care to be taken when composing the output from an amplifier circuit motif with downstream circuit elements, as we outline below.

We implemented this circuit motif in the buffered strand displacement gate framework introduced in Section 2, and simulated it using the beta version of the DSD compiler that includes support for mixing events [16]. These simulation results are presented in Figure 2(b), and show that the amplifier produces correct results for several gain settings, and that the gain of the amplifier can be dynamically adjusted by directly adding more unbuffering strands between amplification reactions.

4 A strand displacement learning circuit

In this section, we will present a buffered strand displacement system that can learn linear functions f of the form

$$f(x_1, x_2) = w_1 \times x_1 + w_2 \times x_2,$$

where w_1 and w_2 are real-valued coefficients and x_1 and x_2 are real-valued inputs. In the current paper, we restrict ourselves to the two-input case for clarity, although the circuit design motifs that we will present could be replicated to handle more inputs. In particular, a bias term could be included by incorporating an additional input signal x_0 that is always supplied with the input value $x_0 = -1$ in each training round. This is standard practice in studies of artificial neural networks. We will present a strand displacement system that learns functions of this form using a stochastic gradient descent algorithm [17]. Gradient descent is a general solution for many optimization tasks, in which the current weight approximations are adjusted to minimize the squared error over the entire training set in each training round. Stochastic gradient descent is a simplification of gradient descent that only considers a single training instance in each training round, making it more amenable to implementation in a molecular computing system.

A molecular computing system that solves this problem must accept the input values x_1 and x_2, compute the predicted output $y = \widehat{w_1} \times x_1 + \widehat{w_2} \times x_2$ based on the current stored weight approximations $\widehat{w_1}$ and $\widehat{w_2}$, compare y with the supplied expected value $d = w_1 \times x_1 + w_2 \times x_2$, and update the stored weight approximations according to the gradient descent weight update rule:

$$\widehat{w_i} := \widehat{w_i} + \alpha \times (d - y) \times x_i, \tag{1}$$

where α is a (small) positive coefficient called the "learning rate". We now present a strand displacement system that implements this learning algorithm using buffered DNA strand displacement gates. Our design can be divided into two subcircuits, as follows.

Predictor subcircuit. The predictor subcircuit is based on the adaptive amplifier presented in Section 3. The input signals, and all other numeric signals, are represented in a dual rail format with a differential encoding, that is, the input signal x_i actually consists of two signals ${x_i}^+$ and ${x_i}^-$, and the value of x_i is interpreted as the concentration difference $[{x_i}^+] - [{x_i}^-]$. The predictor subcircuit design is presented in Figure 3: the initial circuit motif is replicated for each input x_i. Here, and henceforth, we will omit the identities of buffer species from figures if their identity is not important when describing the operation of the circuit. The species highlighted in grey (${x_i}^+$, ${x_i}^-$, d^+, and d^-) are provided by the user to initiate each training round: their concentrations represent the input value x_i and the expected result d that the user must derive using the target weight values. (If $d = 0$ then we must add equal concentrations of d^+ and d^-. Any non-zero concentration is acceptable, as these species must be present to drive the execution of the predictor subcircuit.)

Each positive input signal ${x_i}^+$ is "copied" by a buffered fork gate that generates four signals with the same overall concentration as the original: two of these, ${x_{i1}}^+$ and ${x_{i2}}^+$, are for use by the predictor subcircuit and the remaining two, ${k_{i1}}^+$ and ${k_{i2}}^+$, are for use by the feedback subcircuit (as detailed below). Each negative input signal ${x_i}^-$ is copied similarly.

The key parts of the predictor subcircuit are the buffered strand displacement amplifier motifs. The initial gains of the predictor subcircuit amplifiers encode the initial approximation of each weight value stored in the system. There is one pair of amplifiers per positive input signal and one pair per negative input signal. In each of these pairs, the gain of one amplifier represents the positive component $\widehat{{w_i}^+}$ of the corresponding weight approximation $\widehat{w}_i$, and the gain of the other amplifier represents the negative component $\widehat{{w_i}^-}$. Thus, the positive component of each input is multiplied by both the positive and negative components of the corresponding weight, and similarly for the negative component. The outputs of the predictor subcircuit amplifiers are two species y^+ and y^-, which represent positive and negative components of the current prediction, based on the current input values and stored weight approximations. The amplifier gates are constructed such that the sign of the output species is correct with respect to the signs of the input component and the weight component in each case.

To complete the execution of the predictor subcircuit, the y^+ and y^- species, whose concentrations represent the current prediction, interact with the d^+ and d^- species, whose concentrations represent the expected value of the target function. These species interact via the four buffered reactions shown in the box on the right-hand side of Figure 3, which collectively have the effect of subtracting the value of y from the value of d. We implement this operation via four two-input two-output reactions, in which the $d^\pm$ species catalyze conversion of the $y^\pm$ species to $d^\pm$, with signs chosen such that the resulting concentrations of $d^\pm$ represent the result of a subtraction. We choose to implement subtraction in this way, rather than using annihilator gates that degrade the positive and negative variants of the species, to avoid sequestration of the remaining

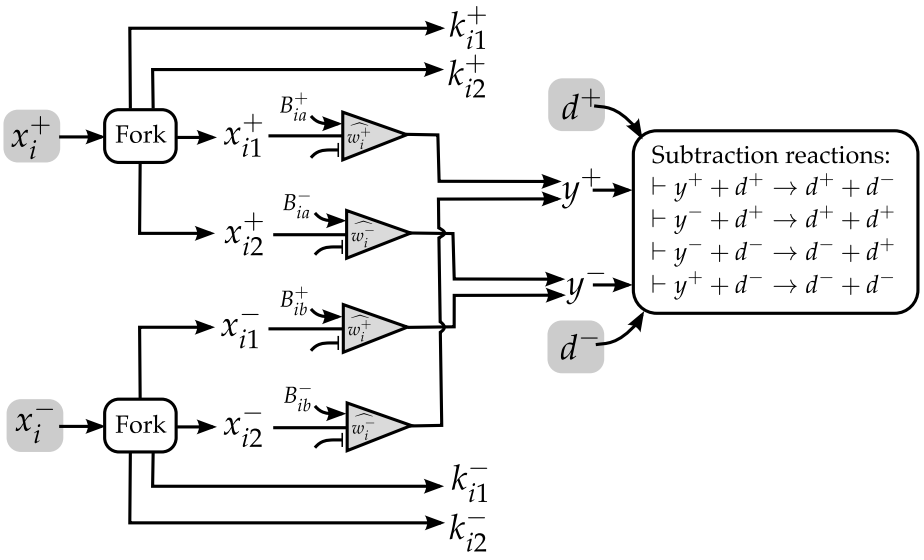

Fig. 3. Design schematic for the predictor subcircuit. The concentrations of the input species ${x_i}^+$ and ${x_i}^-$ (for each input signal x_i) are copied by buffered fork gates to produce species that serve as inputs to the buffered amplifier motifs that implement that linear function prediction, and additional species (${k_{i1}}^\pm, {k_{i2}}^\pm$) that will be used in the feedback subcircuit. The gains of these amplifiers store the current weight approximations. The amplifiers produce species y^+ and y^-, such that $[y^+]-[y^-]$ equals the predicted output value. These species then interact with the d^+ and d^- species, which encode the expected output value, via four buffered reactions that implement subtraction. When the predictor subcircuit reaches steady state, the concentration difference $[d^+]-[d^-]$ should equal $d-y$, i.e., the error in the prediction when compared with the expected output value.

species by the annihilator gates, which has been problematic in other work [16]. For this reason, it is crucial that the $y^\pm$ species are the first input strands consumed by these reaction gates.

Thus, when the predictor subcircuit reactions reach steady state, the concentration difference $[d^+]-[d^-]$ represents the value of $d-y$. If this value is positive, i.e., if $[d^+] > [d^-]$, then the predicted output value was too small. Similarly, if this value is negative, i.e., if $[d^+] < [d^-]$, then the predicted output was too large. The goal of the learning process is to adjust the stored weight approximations $\widehat{w}_i$ to match the target weight approximations, so that $[d^+] = [d^-]$ when the predictor subcircuit reactions reach steady state.

Feedback subcircuit. Once the predictor subcircuit has computed the discrepancy between the predicted function output and the expected output, the feedback subcircuit must use the value of this discrepancy to update the weight approximations stored in

the predictor subcircuit, according to the gradient descent learning rule (1). A design schematic for our feedback subcircuit is presented in Figure 4.

The first point to note from (1) is that the feedback subcircuit must take the concentration of $d^{\pm}$ that denotes the discrepancy from the predictor subcircuit and, for each input, multiply the discrepancy value by the corresponding input value, both of which are concentrations, and by the learning rate constant α. A single buffered amplifier can only multiply an input concentration by a gain factor encoded as a ratio of concentrations. To enable two input concentrations to be multiplied together, we have developed a two-concentration multiplier circuit motif, shown in Figure 4(a). In this motif, we assume that no unbuffering strands are initially present for the uppermost amplifier, which will accept the input signal x and produce the output species. The input signal k activates an amplifier that produces the unbuffering strand for the catalyst gate from the output-producing amplifier, with gain β. An additional input signal, whose value is constant 1, activates an amplifier that produces the unbuffering strand for the degradation gate from the output-producing amplifier, with gain 1. Thus, these secondary amplifiers preset the gain of the output-producing amplifier to be $\beta \times [k]$, and upon addition of the input signal x the resulting concentration of both output species (y and z) will be $\beta \times [x] \times [k]$, as desired. (Here, we include two outputs because this variation is called for in our two-input learning circuit design.)

The feedback subcircuit uses the two-concentration multiplier circuit motif extensively, to implement the weight update rule (1). Figure 4(b),(c) presents the feedback subcircuit design for the two-input case. Execution of the feedback subcircuit is initialized by buffered fork gates that copy the $d^{\pm}$ species into four species ${d_a}^{\pm}$, ${d_b}^{\pm}$, ${d_c}^{\pm}$, and ${d_d}^{\pm}$. We assume that there are initially no unbuffering strands for these fork gates and, once the predictor subcircuit has run to completion, the addition of these unbuffering strands triggers execution of the feedback subcircuit.

For each combination of signs for the leftover $d^{\pm}$ species and the copied input species ${k_{ij}}^{\pm}$ from the predictor subcircuit, the copied $d^{\pm}$ species and the copied ${k_{ij}}^{\pm}$ species serve as inputs to an instance of the two-concentration multiplier circuit motif. As described above, this circuit motif multiplies these values together (and by a scaling factor $\beta = \alpha \times \delta$, where α is the learning rate constant and δ is the denominators of the weight ratios in the predictor subcircuit) and generates additional unbuffering strands for certain amplifier gates from the predictor subcircuit. From the two-input predictor subcircuit design from Figure 3, we see that additional unbuffering strands B_{ia}^{+} and B_{ib}^{+} must be generated to increase weight approximation $\widehat{w}_i$, and that additional unbuffering strands B_{ia}^{-} and B_{ib}^{-} must be generated to decrease weight approximation $\widehat{w}_i$. These pairs of species must be generated together, so that the pairs of amplifiers from the predictor subcircuit that are initialized with the same weight approximation are updated together.

5 Results

We encoded the learning circuit design from Section 4 in the DSD programming language [18] and used the associated DSD compiler, with the "*Infinite*" reaction semantics [10], to generate MATLAB code that implements an ODE model of the system. The initial state of the two-input system consists of 278 species, of which the majority

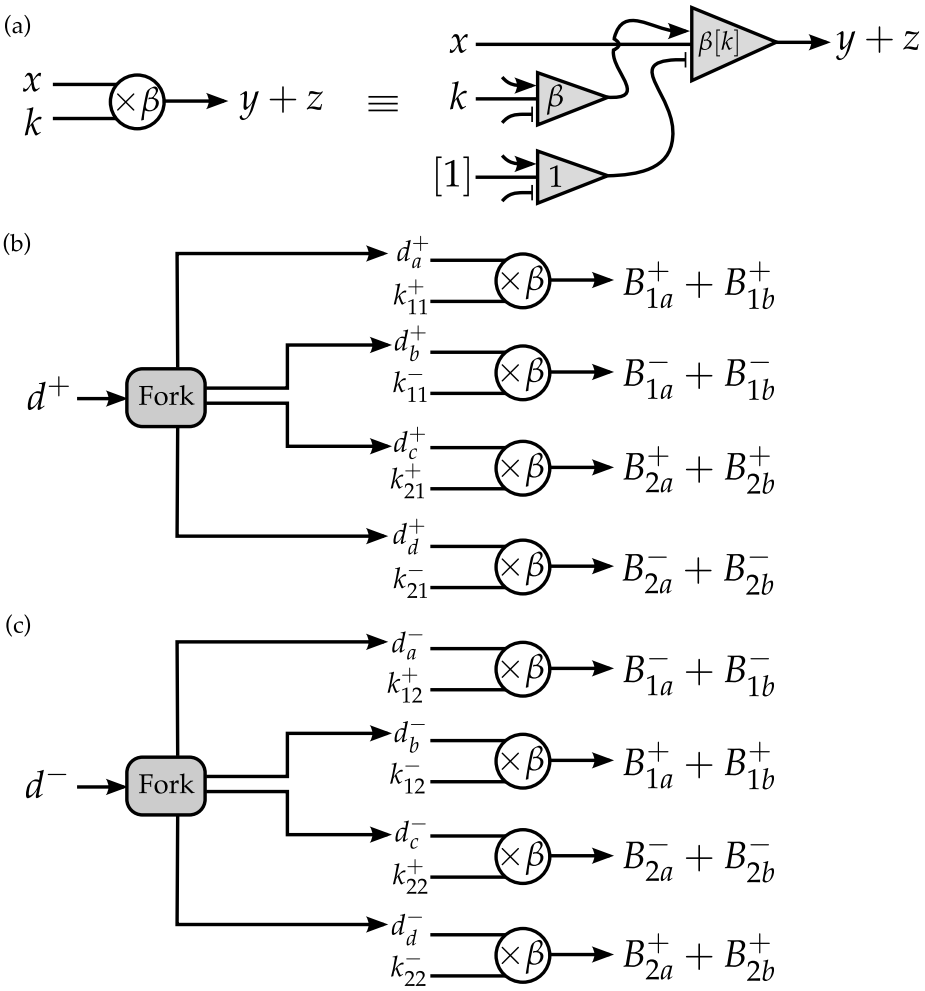

Fig. 4. Feedback subcircuit. **a.** Graphical shorthand for a multi-amplifier motif that enables one input concentration ($[x]$) to be multiplied by another concentration ($[k]$) and a scalar scaling factor (β). **b.** and **c.** Design schematic for the feedback subcircuit. Here, the scaling factor $\beta = \alpha \times \delta$, where α is the learning rate constant and δ is the denominators of the weight ratios in the predictor subcircuit. The feedback circuitry from **b.** is activated when d^+ is left over from the predictor subcircuit, and the feedback circuitry from **c.** is activated when d^- is left over from the predictor subcircuit.

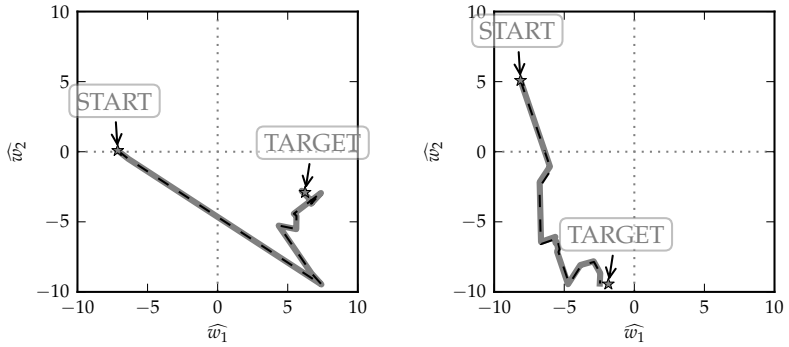

Fig. 5. Two example weight traces from DNA learning circuit simulations (grey solid lines), overlaid with corresponding traces from our Python reference implementation (black broken lines). The lines coincide closely in all simulations, not just the examples shown here, indicating that the DNA circuit works correctly.

(222) are gate complexes. It is worth noting, however, that many of these species are variants with different combinations of history domains, so the design complexity of the circuit is not as high as suggested by the raw species counts. Furthermore, the number of species should scale linearly with the number of input signals, so the circuit design presented here could be replicated to learn similar functions of more than two inputs.

We simulated the ODE model of the learning circuit using a custom MATLAB simulation routine that invokes a stiff ODE solver and that also allows mixing events to be executed at certain time points during the simulation. These mixing events simulate the addition of inputs to the system, or the removal of species, by the experimenter at certain time points. In principle, stochastic simulations could also be used to investigate the behavior of the system in the limit of low species populations, though the requirement for high populations of buffered gates could lead to poor performance in a stochastic simulation of the full system. We found that the size of the buffer, that is, the quantity of unbuffered gates waiting to be activated, did not affect the accuracy of the computation performed by the learning circuit. It did, however, reduce the number of training rounds that could be conducted before the buffer was depleted, at which point no further training could be carried out. This point is discussed in Section 6 below.

The initial state of the system consists of the various buffered gates and their unbuffering strands (with the exceptions of the unbuffering strands for the fork gates and output-generating amplifiers in the feedback subcircuit, as described above). The initial weight approximations $\widehat{w}_i$ are encoded as the gain settings of the amplifiers in the predictor subcircuit. After the gates have unbuffered (after 500 s), the first training inputs are added, which consist of the $x_i^{\pm}$ species and the $d^{\pm}$ species, whose concentrations encode the input values and the expected function output, respectively. At the same time, we also add the constant-valued signals that serve as an input to the feedback subcircuit, so that the output-generating amplifiers from the feedback subcircuit will be primed with the correct gain values before it starts executing. After a further 2000 s,

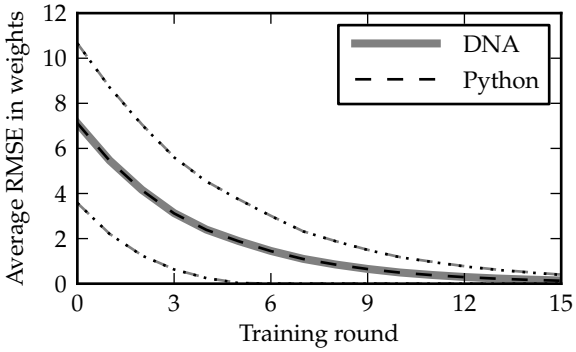

Fig. 6. Learning curves from DNA learning circuit simulations and our Python reference implementation. The average RMSE in the weight approximations was computed after each training round for 1,000 training schedules, each 15 rounds in length. Non-solid lines are one standard deviation above and below the mean. Again, the lines coincide very well, indicating that our DNA circuit design works correctly.

when the predictor subcircuit has completed its execution, we add unbuffering strands for the fork gates from the feedback subcircuit, which triggers execution of the feedback subcircuit. After a further 3500 s, when the feedback subcircuit has finished updating the weight approximations stored in the predictor subcircuit, we set the concentrations of any remaining unbuffered (active) fork gates and output-generating gates in the feedback subcircuit to zero, to reset the state of the feedback subcircuit. We also add the second set of training inputs at this time, and iterate until the specified sequence of training instances have all been presented.

We ran a total of 1,000 simulations, with initial and target weight values and input values selected from a uniform random distribution over the interval $[-10, 10]$ and with a fixed learning rate value $\alpha = 0.01$. Figure 5 shows simulation results illustrating how the weight values evolve over the course of two 15-round example training schedules. In the two-input case, we can plot the weight space as a 2D plot and observe the trajectory of how the weights evolve from their initial values (labeled "START") towards the true values (labeled "TARGET") over the training period. In each example, the trajectory for the simulation of the DNA strand displacement learning circuit is overlaid with a trajectory derived from a reference implementation of the stochastic gradient descent learning rule (1) in Python, using the same initial state and the same training schedule. The weight trajectories shown in Figure 5 are representative of the agreement between the DNA system and the reference implementation observed in all cases, and of the fact that the weight trajectories correctly home in on the target weight values. These results give us confidence in the correct operation of our circuit design as an implementation of the learning rule (1).

Furthermore, to investigate the aggregate learning performance from our simulations, we computed the root mean square error (RMSE) of the stored weight approximations compared with the target weight values at each step of each of the 1,000 train-

ing simulations. The average RMSE values over the 1,000 training simulations, and the standard deviations, are plotted in Figure 6. Again, the results from the DNA and Python versions of the learning system are overlaid precisely, indicating that the DNA circuit follows the expected behavior. The RMSE in the weight approximations is almost zero after just 15 training rounds, suggesting that an experimental implementation of this system could be trained to a reasonable degree of accuracy in a limited timeframe.

6 Discussion

We have presented a design for a DNA learning circuit based on buffered DNA strand displacement reactions, and demonstrated via simulation results that the design works as intended and can learn target weight values in a reasonable timeframe. This feedback subcircuit design surpasses our previous work in this area [5], by allowing negative weight values and input values. It also alleviates the problems we saw in [5] with poor performance when trying to learn weight values near zero, by implementing a weight update rule that is symmetric in the positive and negative directions. The weight update rule used in this work (1) is a well-studied gradient descent learning scheme, whose performance has been studied extensively [17]. It is well known that the learning rate parameter (α) can have a significant effect on the rate of convergence of the learning process. Indeed, many different procedures for reducing the learning rate over time to achieve rapid convergence have been studied, and our system should respond to such adjustments in the same way as the reference algorithm that we implemented in Python.

Our circuit was specifically designed to avoid the issue of input sequestration, which can be problematic in DNA strand displacement systems. Soloveichik's compilation scheme for abstract chemical reactions [8] dealt with this issue by adding additional reaction gates to compensate by sequestering all other species similarly, which slows down the system. Other work [16] has dealt with input sequestration by artificially increasing the values of certain input signals. This consideration led to the design of the "subtractor" circuit motif shown in Figure 3 which, while elegant, has the property that the absolute values of the positive and negative components of the weight approximations and other signals processed by the system grow monotonically over the course of a multi-round training session, draining other species out of the system. This problem could be ameliorated by the inclusion of an annihilator gate in the feedback subcircuit that takes d^+ and d^- as inputs but produces no output, which could reduce the absolute sizes of the signals. This annihilator gate would only need to compete with the rest of the feedback subcircuit to drain some of the excess $d^\pm$ signals form the system, and we will explore this alternative in future work.

Our use of buffered gates to implement adaptive strand displacement circuits offers a route to implement systems whose operation can be extended by the replenishment of buffered gate species when they are depleted, without adversely affecting the kinetics of the system. This is an important consideration for the implementation of molecular learning circuits. Furthermore, in this paper we have implemented the stochastic gradient descent weight update rule in conjunction with a linear transfer function, which allows the circuit to learn linear classification functions. However, the feedback subcircuit design presented here could be used to learn other functional forms, including non-

linear functions, by simply replacing the predictor subcircuit to compute the desired function of the provided training inputs. A major challenge is to implement other transfer functions, in particular, non-linear transfer functions such as the Heaviside function, which is used in classical expositions of perceptron learning [19]. Specifically, the challenge here is to implement a reusable circuit that can amplify its output to a fixed level. Finally, to build molecular systems that can learn arbitrary functions it would be necessary to connect a number of such units into networks to be trained by backpropagation, which could be achieved by cascading several of the circuit motifs described here.

Acknowledgments. This material is based upon work supported by the National Science Foundation under grants 1318833 and 1422840. M.R.L. gratefully acknowledges support from the New Mexico Cancer Nanoscience and Microsystems Training Center.

References

1. D. M. Morens and A. S. Fauci. Emerging infectious diseases: Threats to human health and global stability. *PLOS Pathogens*, 9(7):e1003467, 2013.
2. D. Y. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, 3(2):103–113, 2011.
3. L. Qian and E. Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface*, 8(62):1281–1297, 2011.
4. L. Qian, E. Winfree, and J. Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475:368–372, 2011.
5. M. R. Lakin, A. Minnich, T. Lane, and D. Stefanovic. Design of a biochemical circuit motif for learning linear functions. *Journal of the Royal Society Interface*, 11(101):20140902, 2014.
6. P. Banda, C. Teuscher, and M. R. Lakin. Online learning in a chemical perceptron. *Artificial Life*, 19(2):195–219, 2013.
7. P. Banda, C. Teuscher, and D. Stefanovic. Training an asymmetric signal perceptron through reinforcement in an artificial chemistry. *Journal of the Royal Society Interface*, 11:20131100, 2014.
8. D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences USA*, 107(12):5393–5398, 2010.
9. L. Cardelli. Strand algebras for DNA computing. *Natural Computing*, 10(1):407–428, 2010.
10. M. R. Lakin, S. Youssef, L. Cardelli, and A. Phillips. Abstractions for DNA circuit design. *Journal of the Royal Society Interface*, 9(68):470–486, 2012.
11. A. J. Genot, D. Y. Zhang, J. Bath, and A. J. Turberfield. Remote toehold: a mechanism for flexible control of DNA hybridization kinetics. *Journal of the American Chemical Soceity*, 133:2177–2182, 2011.
12. D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, 318:1121–1125, 2007.
13. L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.
14. D. Y. Zhang and G. Seelig. DNA-based fixed gain amplifiers and linear classifier circuits. In *Proceedings of the 16th International Conference on DNA Computing and Molecular Programming*, volume 6518 of *Lecture Notes in Computer Science*, pages 176–186. Springer-Verlag, 2011.
15. K. Oishi and E. Klavins. Biomolecular implementation of linear I/O systems. *IET Systems Biology*, 5(4):252–260, 2011.

16. B. Yordanov, J. Kim, R. L. Petersen, A. Shudy, V. V. Kulkarni, and A. Phillips. Computational design of nucleic acid feedback control circuits. *ACS Synthetic Biology*, 3(8):600–616, 2014.
17. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, second edition, 2001.
18. M. R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.
19. M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA, second edition, 1972.