Implementation of Support Vector Machines using Chemical Reaction Networks

Amey Choudhary[1], Abhishek Deshpande[2], and Jiaxin Jin[3]

[1,2]Center for Computational Natural Sciences and Bioinformatics, International Institute of Information Technology, Hyderabad
[3]Department of Mathematics, University of Louisiana at Lafayette

April 2, 2026

Abstract

Can machine learning algorithms be implemented using chemistry? We demonstrate that this is possible in the case of support vector machines (SVMs). SVMs are powerful tools for data classification, leveraging Vapnik–Chervonenkis theory to handle high-dimensional data and small datasets effectively. In this work, we propose a chemical reaction network scheme for implementing SVMs, utilizing the steady-state behavior of reaction network dynamics to model key computational aspects of SVMs. This approach introduces a novel biochemical framework for implementing machine learning algorithms in non-traditional computational environments.

1 Introduction

Living cells respond to external stimuli through mechanisms, such as gene regulatory networks. Chemical reaction network theory hopes to construct complex circuits, such as oscillators and switches, from simple building blocks. Recently, these designs have enabled the development of complicated biological circuits for real-world applications. Our work aims to enable computation in living environments, where modern computers are currently unable to operate.

Chemical reaction networks provide a robust framework for biomolecular interactions and can be realized through DNA strand displacement reactions [19], providing a physical basis in cellular environments. This makes them a powerful "programming language" for molecular computation, benefiting from DNA's sequence specificity and high-density information storage. However, programming reaction networks for more sophisticated tasks, such as training and testing, remains a significant challenge.

The idea of using reaction networks for performing computation dates back to 1994, when Adleman solved a seven-node Hamiltonian path problem using DNA strands [1].

Winfree and Qian later developed DNA strand displacement schemes capable of realizing increasingly complex molecular circuits [5, 18]. More recently, reaction networks have been used to implement various machine learning algorithms. In particular, Stojanovic et al.[14] constructed reaction networks for automating decision tree implementations, while Gopalkrishnan et al.[9, 24, 25] designed schemes for maximum likelihood estimation and Boltzmann machine simulation [15, 16, 17]. Reaction network implementations of neural network dynamics have also been demonstrated [2, 7, 12].

In this paper, our focus is on *Support Vector Machines (SVMs)*. Introduced by Vapnik and Chervonenkis in the 1960s, SVMs are supervised classification algorithms. The essential idea of SVM is to construct a hyperplane that maximally separates data points from different classes, creating the largest possible margin. While SVMs are highly effective for linearly separable data, nonlinear classification requires the use of *kernel functions* to map data onto higher-dimensional spaces. This is also called the *kernel trick* for SVM [11].

We design a reaction network scheme to implement a soft-Margin SVM, leveraging biochemical processes for machine learning tasks. Our reaction network-based model is capable of (1) loading input data points in batches, (2) performing inference, (3) backpropagating the classification loss, and (4) updating weights via gradient descent. To ensure these operations are executed sequentially, we employ oscillating molecular species to achieve time-scale separation. We validate our design by simulating the reaction network on synthetic datasets using Python and demonstrate effective convergence, with weight updates closely matching those obtained from standard SVM implementations. This work introduces a novel biochemical framework for executing machine learning algorithms in non-traditional computational environments, broadening the scope of reaction network applications in synthetic biology.

Structure of the paper: The paper is organized as follows. In Section 2, we introduce reaction networks and recall their properties. Section 3 introduces support vector machines and describes their functioning as an optimization problem. In Section 4, we introduce the concept of dual-rail encoding to handle negative weights and biases. Further, we describe basic modules like addition, subtraction, multiplication, comparison, and approximate majority that will be used as building blocks in the reaction network that simulates the SVM. In section 5, we describe our implementation of our molecular clock using *Hopf* reactions. In Section 6, we describe the assignment module that is used for loading training in batches. Sections 7 and 8 describe the reaction network that simulates the feedforward and backpropagation components of the SVM, respectively. In Section 9, we present and validate the results for our reaction network and compare the values of weights and biases obtained using reaction networks and those obtained using the traditional SVM implementations. To validate these results, we also compare their trajectories as a function of time (epochs in our case) and find that they are in close agreement. Section 10 summarizes our contributions and outlines directions for future work.

2 Chemical Reaction Networks

Definition 2.1 ([8])**.** A Chemical Reaction Network (reaction network) is a triple $(\mathcal{S}, \mathcal{C}, \mathcal{R})$ consisting of a finite set of *species* $\mathcal{S} = \{X_1, \ldots, X_n\}$, a set of *complexes* $\mathcal{C} = \{C_1, \ldots, C_s\}$, and a set of *reactions* $\mathcal{R} = \{R_1, \ldots, R_r\}$. Each reaction describes an interaction between species given by:

$$R_j : \sum_{i=1}^{n} \alpha_{ij} X_i \to \sum_{i=1}^{n} \beta_{ij} X_i \quad \text{with } j = 1, \ldots, r,$$

where the left-hand side (*reactant complex*) and the right-hand side (*product complex*) are linear combinations of species. For each reaction R_j, the non-negative integer coefficients α_{ij} and β_{ij} represent the *stoichiometric coefficients* of species X_i in the reactant and product complexes, respectively.

Example 1. Consider the reaction network $(\mathcal{S}, \mathcal{C}, \mathcal{R})$ shown in Figure 1. It consists of three species:

$$\mathcal{S} = \{X_1, X_2, X_3\},$$

three complexes:

$$\mathcal{C} = \{X_1 + X_2,\ 2X_2,\ X_3\},$$

and four reactions:

$$\mathcal{R} = \{X_1 + X_2 \to X_3, \quad 2X_2 \to X_1 + X_2, \quad 2X_2 \to X_3, \quad X_3 \to X_1 + X_2\}.$$

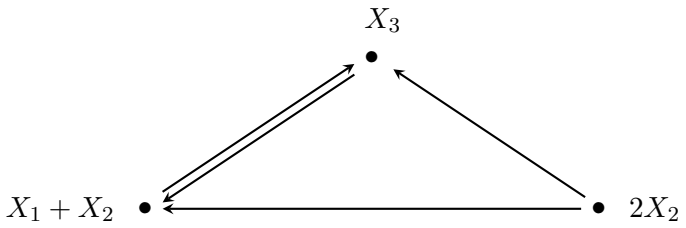


Figure 1: A reaction network $(\mathcal{S}, \mathcal{C}, \mathcal{R})$ with three species, three complexes, and four reactions.

One of the most widely used models in reaction network studies is based on *mass-action kinetics* [10]. In this framework, the reaction rate is determined by the product of the reactant concentrations, each raised to its corresponding stoichiometric coefficient. The detailed definition is provided below.

Definition 2.2 ([8])**.** Given a reaction network $(\mathcal{S}, \mathcal{C}, \mathcal{R})$, each reaction R_j is assigned a positive constant k_j (*reaction rate constant*) for $1 \leq j \leq r$. Let $\boldsymbol{k} = (k_1, \ldots, k_r) \in \mathbb{R}^r_{>0}$ denote the *reaction rate vector*. The *mass-action system* generated by $(\mathcal{S}, \mathcal{C}, \mathcal{R}, \boldsymbol{k})$ is given by:

$$\frac{dx_i(t)}{dt} = \sum_{j=1}^{r} k_j \prod_{l=1}^{n} x_l^{\alpha_{lj}} (\beta_{ij} - \alpha_{ij}) \quad \text{with } i = 1, \ldots, n,$$

where $\boldsymbol{x}(t) = (x_1(t), \ldots, x_n(t))$ represents the concentrations of species $X_1, \ldots, X_n$ at time t.

For example, recall the reaction network from Figure 1 (see Example 1). Given the reaction rate vector $\boldsymbol{k} = (k_1, \ldots, k_4)$, the reaction rates are assigned as follows:

$$X_1 + X_2 \xrightarrow{k_1} X_3 \qquad 2X_2 \xrightarrow{k_2} X_1 + X_2 \qquad 2X_2 \xrightarrow{k_3} X_3 \qquad X_3 \xrightarrow{k_4} X_1 + X_2$$

Under mass-action kinetics, the associated dynamical system is

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} &= k_1 x_1 x_2 \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix} + k_2 x_2^2 \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} + k_3 x_2^2 \begin{pmatrix} 0 \\ -2 \\ 1 \end{pmatrix} + k_4 x_3 \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \\ &= \begin{pmatrix} -k_1 x_1 x_2 + k_2 x_2^2 + k_4 x_3 \\ -k_1 x_1 x_2 + (k_2 - 2k_3) x_2^2 + k_4 x_3 \\ k_1 x_1 x_2 + k_3 x_2^2 - k_4 x_3 \end{pmatrix}. \end{aligned}$$

Reaction networks offer a robust framework for implementing molecular-level computations since the mass-action systems generated by them have polynomial right-hand sides.

In the following, we introduce two key concepts in reaction networks: catalytic and non-catalytic species, which will be frequently used in Section 4.

Definition 2.3 ([8])**.** Let $(\mathcal{S}, \mathcal{C}, \mathcal{R})$ be a reaction network with reactions $R_1, \ldots, R_r \in \mathcal{R}$, where each reaction has the form

$$R_j : \sum_{i=1}^{n} \alpha_{ij} X_i \to \sum_{i=1}^{n} \beta_{ij} X_i$$

(a) A species $X_i \in \mathcal{S}$ is said to be *catalytic* if the number of species X_i in the reactant and product complexes remains the same for every reaction. Specifically, X_i is a catalytic species if and only if $\alpha_{ij} = \beta_{ij}$ for every $1 \leq j \leq r$.

(b) A species $X_i \in \mathcal{S}$ is said to be a *non-catalytic* species if it is not catalytic, that is, there exists a reaction R_j such that $\alpha_{ij} \neq \beta_{ij}$.

Example 2. Consider the following reaction network:

$$\emptyset \to X_1 + X_2 \qquad X_1 + X_3 \to X_3$$

In this example, the reaction network consists of three species, $\mathcal{S} = \{X_1, X_2, X_3\}$. Among them, X_1 and X_2 are non-catalytic, while X_3 is a catalytic species.

3 Support Vector Machines

Support Vector Machines (SVMs) [6, 21] are supervised machine learning algorithms. SVMs have attracted widespread attention in machine learning, data mining, and pattern recognition tasks. In particular, they are particularly useful in binary classification problems.

SVM works on the principle of generating a surface that separates data into classes by maximizing the distance between itself and the data points. In particular, SVM assigns the data points labels by finding the *optimal hyperplane*. This hyperplane corresponds to one that is at the maximum distance from the closest point of any class. Given a new data point, it can be classified and given labels according to which side of the hyperplane it will be in feature space.

If the data cannot be separated by such a hyperplane, then the SVM finds the hyperplane as in the hard margin case, with the caveat that a penalty is added each time a point crosses the margin. These SVMs are referred to as soft margin SVMs.

3.1 Hard-Margin SVM Formulation

Given a set of data points (which are feature vectors) that belong to either of the two classes, one needs to decide to which of the two classes a new data point belongs. More formally, given a dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ are the feature vectors and $y^{(i)} \in \{1, -1\}$ are the binary class labels, the goal is to find a weight vector $\mathbf{w} \in \mathbb{R}^d$ such that it maximizes the distance of its nearest point for each class and bias $b \in \mathbb{R}$ such that:

$$y^{(i)}(\mathbf{w} \cdot \boldsymbol{x}^{(i)} + b) \geq 1 \quad \text{for all } 1 \leq i \leq N. \tag{1}$$

The optimization problem for a *hard-margin SVM* is defined as:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \tag{2}$$

subject to the constraint that all training points satisfy Equation (1). This objective aims to maximize the margin, which is inversely proportional to $\|\mathbf{w}\|$.

Figure 2 shows a hard-margin SVM for a two-dimensional input and class label taking values -1 and 1.

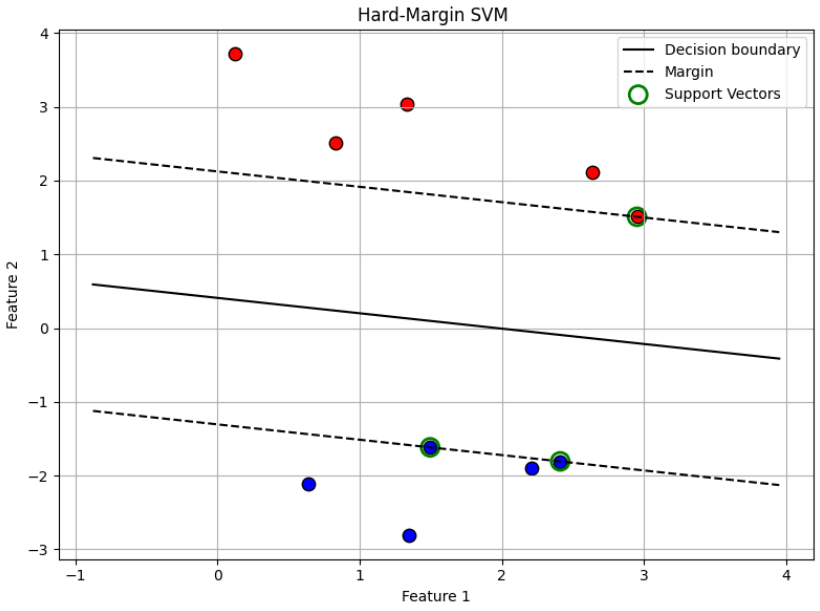


Figure 2: Visualization of Hard Margin SVM for two-dimensional input and class label $\in \{1, -1\}$; red points denote class 1 and blue points denote class -1.

3.2 Theoretical Justification

While the hard-margin SVM provides an ideal formulation for linearly separable data, directly solving this constrained optimization problem can be complex. In practice, we adopt an unconstrained formulation using *hinge loss* and *L2 regularization*, which corresponds to a *soft-margin SVM*:

$$\min_{\mathbf{w},b} \quad \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} \max(0, 1 - y^{(i)}(\mathbf{w} \cdot \boldsymbol{x}^{(i)} + b)),$$

where $\lambda > 0$ is the regularization parameter. This soft-margin approach is more robust as it allows for margin violations, penalizing them through the hinge loss. A key insight is that when the data is linearly separable, the hinge loss term becomes zero for all training points, effectively reducing the soft-margin objective back to the original hard-margin problem in Equation (2).

The iterative training algorithm we employ is a form of gradient descent derived directly from this soft-margin SVM formulation. For a single training example $(\boldsymbol{x}, y)$, let $f(\boldsymbol{x}) = \mathbf{w} \cdot \boldsymbol{x} + b$. The hinge loss for this example is

$$\text{Loss}(\mathbf{w}, b) = \max(0, 1 - yf(\boldsymbol{x})).$$

The subgradients of this loss with respect to $\mathbf{w}$ and b are:

$$\frac{\partial \text{Loss}}{\partial \mathbf{w}} = \begin{cases} \lambda \mathbf{w} - y\boldsymbol{x} & \text{if } yf(\boldsymbol{x}) < 1, \\ \lambda \mathbf{w} & \text{otherwise,} \end{cases} \qquad \frac{\partial \text{Loss}}{\partial b} = \begin{cases} -y & \text{if } yf(\boldsymbol{x}) < 1, \\ 0 & \text{otherwise.} \end{cases}$$

These subgradients form the basis for our iterative update rules.

3.3 Iterative Training Algorithm

In this section, we give an iterative procedure to train the SVM by adjusting the weight vector $\mathbf{w}$ and the bias b according to an iterative training algorithm.

Given data points $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, a regularization parameter λ, and a learning rate $\eta > 0$, the weight $\mathbf{w}$ and the bias b are updated as follows:

1. Initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$.
2. For each data point $(\boldsymbol{x}^{(i)}, y^{(i)})$ in the dataset:
   - Compute the decision function: $f(\boldsymbol{x}^{(i)}) = \mathbf{w} \cdot \boldsymbol{x}^{(i)} + b$.
   - If there is a misclassification or margin violation ($y^{(i)} f(\boldsymbol{x}^{(i)}) < 1$), then
   $$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta(\lambda \mathbf{w} - y^{(i)} \boldsymbol{x}^{(i)}) \\ b &\leftarrow b + \eta y^{(i)} \end{aligned} \tag{3}$$
   - If there is no misclassification ($y^{(i)} f(x^{(i)}) \geq 1$), then
   $$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \lambda \mathbf{w} \\ b &\leftarrow b \end{aligned} \tag{4}$$
3. Repeat this process for multiple epochs until convergence or a set number of iterations is reached.

This iterative process adjusts $\mathbf{w}$ and b in the direction that minimizes the regularized hinge loss, effectively moving the hyperplane to reduce errors for misclassified points. The learning rate η controls the step size of these updates, influencing the convergence rate. In the separable case, these updates converge toward the maximum-margin hyperplane of SVMs. This approach provides a computationally efficient method for training SVMs, particularly beneficial in online or large-scale learning scenarios.

To implement batch processing in SVM, we modify the update rule. The modification applies specifically to Equation (3) and (4). The input-dependent parameter updates are aggregated across a batch size $n_{\text{batch-size}}$, and then we perform a single update. For each sample in the batch, we define

$$w_{\text{upd}}^{(i)} = \begin{cases} \eta y^{(i)} \boldsymbol{x}^{(i)}, & \text{if } y^{(i)} f(\boldsymbol{x}^{(i)}) < 1, \\ 0, & \text{otherwise,} \end{cases} \qquad b_{\text{upd}}^{(i)} = \begin{cases} \eta y^{(i)}, & \text{if } y^{(i)} f(\boldsymbol{x}^{(i)}) < 1, \\ 0, & \text{otherwise.} \end{cases}$$

The batch update is given by

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta\lambda\mathbf{w} + \frac{1}{n_{\text{batch-size}}} \sum_{i=1}^{n_{\text{batch-size}}} w_{\text{upd}}^{(i)}, \\ b &\leftarrow b + \frac{1}{n_{\text{batch-size}}} \sum_{i=1}^{n_{\text{batch-size}}} b_{\text{upd}}^{(i)}. \end{aligned} \tag{5}$$

3.4 Decision Rule

After training, the classification decision for a new data point $\hat{\boldsymbol{x}} \in \mathbb{R}^d$ is determined by

$$f(\hat{\boldsymbol{x}}) = \mathbf{w} \cdot \hat{\boldsymbol{x}} + b,$$

and the predicted class is given by

$$\hat{y} = \begin{cases} 1 & \text{if } f(\hat{\boldsymbol{x}}) \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

This iterative approach, while not the standard quadratic programming formulation of SVM, provides a computationally efficient method for training SVMs and is particularly useful in online or large-scale learning scenarios.

4 Operation Modules

In this section, we develop the reaction network scheme for the individual submodules that will be used for implementing the *feedforward* and *backpropagation* steps in Sections 7 and 8. To this end, we note that when dealing with weights and biases, negative values may arise, and a mechanism is required to accommodate them. This issue is addressed in the next subsection on *dual-rail encoding*.

4.1 Dual-rail Encoding

In a physical setting, the concentration of species is always non-negative. As data points and weights can be negative, we use the *dual-rail encoding* [22]. Specifically, for any variable $\zeta \in \mathbb{R}$, we take two species ζ^+ and ζ^- with non-negative concentrations $\zeta^+(t) \geq 0$ and $\zeta^-(t) \geq 0$. For all $t \geq 0$, the difference between $\zeta^+(t) - \zeta^-(t)$ represents the real value ζ.

Here, we list some basic modules from [23]. that will be used in the implementation of the feedforward and backpropagation modules. We begin by introducing notation that will be used throughout the remainder of this paper.

Notation: We denote by $[X(t)]$ the concentration of species X at time t. If X is a catalyst, whose concentration remains constant for all $t \geq 0$, we write $[X]$. Finally, $[X^{ss}]$ denotes the concentration of X at steady state.

4.2 Addition Module

In order to add two numbers, we consider the following network:

$$\begin{aligned} A &\xrightarrow{1} A + C \\ B &\xrightarrow{1} B + C \\ C &\xrightarrow{1} \emptyset \end{aligned} \tag{6}$$

In this network, the original input species A and B are catalysts, and thus their concentrations remain constant. The concentrations of species A and B are added, with the resulting value stored in the steady-state concentration of C. Let $[A] = [A(0)]$ and $[B] = [B(0)]$. Then the dynamics associated with the network (6) is given by

$$\frac{d[C(t)]}{dt} = [A] + [B] - [C(t)].$$

The steady-state concentration of C is given by

$$[C^{ss}] = [A] + [B].$$

4.3 Multiplication Module

In order to multiply, we consider the following network:

$$\begin{aligned} A + B &\xrightarrow{1} A + B + C \\ C &\xrightarrow{1} \emptyset \end{aligned} \tag{7}$$

In this network, the species A and B are catalysts. The concentrations of species A and B are multiplied, with the resulting value stored in the steady-state concentration of C. Let $[A] = [A(0)]$ and $[B] = [B(0)]$. Then the dynamics associated with the network (7) is given by

$$\frac{d[C(t)]}{dt} = [A][B] - [C(t)].$$

The steady-state concentration of C is given by

$$[C^{ss}] = [A][B].$$

4.3.1 Dual Rail Encoding in Multiplication Module

We now demonstrate how dual-rail encoding is performed in the multiplication module. Similar mechanisms will hold for other modules in this paper.

Let A_p, A_n and B_p, B_n be the dual-rail species corresponding to signed quantities A and B, respectively, such that

$$A = [A_p] - [A_n], \qquad B = [B_p] - [B_n].$$

The objective is to compute the product $C = A \cdot B$ using dual-rail encoding. Towards this, we have

$$C = A \cdot B = (A_p - A_n) \cdot (B_p - B_n) = A_pB_p + A_nB_n - A_pB_n - A_nB_p = (A_pB_p + A_nB_n) - (A_pB_n + A_nB_p)$$

In the dual-rail form, we introduce species C_p and C_n so that

$$C = [C_p] - [C_n],$$

where

$$[C_p] = A_pB_p + A_nB_n, \qquad [C_n] = A_pB_n + A_nB_p.$$

We implement $[C_p]$ and $[C_n]$ using the multiplication module (7). The corresponding reaction network is given by:

$$\begin{aligned}
A_p + B_p &\to A_p + B_p + C_p \\
A_n + B_n &\to A_n + B_n + C_p \\
A_p + B_n &\to A_p + B_n + C_n \\
A_n + B_p &\to A_n + B_p + C_n \\
C_p &\to \emptyset \\
C_n &\to \emptyset
\end{aligned}$$

The dynamics of the system give

$$[C_p^{ss}] = A_pB_p + A_nB_n, \qquad [C_n^{ss}] = A_pB_n + A_nB_p,$$

thereby correctly encoding the signed product $C = A \cdot B$ in dual-rail form.

4.4 Comparison and Approximate Majority Modules

Our module is designed to compare the concentrations of species and set the corresponding flags. This functionality is achieved using a *comparison module* and an *approximate majority module*, triggered one after another.

Comparison Module

In the comparison module, the inputs X and Y are assigned to flag species X_{gY} and X_{lY} in a normalized fashion. The network for implementing this is given by

$$\begin{aligned} X_{gY} + Y &\xrightarrow{1} X_{lY} + Y \\ X_{lY} + X &\xrightarrow{1} X_{gY} + X \end{aligned} \tag{8}$$

In this network, the species X and Y are catalysts. Let $[X] = [X(0)]$ and $[Y] = [Y(0)]$. To normalize the values within the range $[0, 1]$, we set the initial concentrations of X_{gY} and X_{lY} such that

$$[X_{gY}(0)] + [X_{lY}(0)] = 1.$$

The dynamical system corresponding to the network (8) is given by

$$\begin{aligned} \frac{d[X_{gY}(t)]}{dt} &= -[X_{gY}(t)][Y] + [X_{lY}(t)][X], \\ \frac{d[X_{lY}(t)]}{dt} &= [X_{gY}(t)][Y] - [X_{lY}(t)][X]. \end{aligned}$$

Adding the two equations above gives us that $[X_{gY}(t)] + [X_{lY}(t)]$ is preserved for all time $t \geq 0$. At steady state, we get

$$\frac{[X_{gY}^{ss}]}{[X_{lY}^{ss}]} = \frac{[X]}{[Y]}.$$

Since $[X_{lY}^{ss}] + [X_{gY}^{ss}] = [X_{gY}(0)] + [X_{lY}(0)] = 1$, this implies that

$$[X_{gY}^{ss}] = \frac{[X]}{[X] + [Y]}, \qquad [X_{lY}^{ss}] = \frac{[Y]}{[X] + [Y]}.$$

Moreover, by direct computation, we can derive that this steady state is globally attracting. For example, if $[X] = 60$ and $[Y] = 40$, the flag species X_{gY} and X_{lY} will converge to 0.6 and 0.4, respectively.

Approximate Majority Module

We use the following network ([4]) to implement the approximate majority module:

$$\begin{aligned} X_{gY} + X_{lY} &\xrightarrow{1} X_{lY} + B \\ B + X_{lY} &\xrightarrow{1} X_{lY} + X_{lY} \\ X_{lY} + X_{gY} &\xrightarrow{1} X_{gY} + B \\ B + X_{gY} &\xrightarrow{1} X_{gY} + X_{gY} \end{aligned} \tag{9}$$

where B is an intermediate species.

The dynamical system corresponding to the network (9) is given by

$$\begin{aligned}
\frac{d[X_{gY}(t)]}{dt} &= -[X_{gY}(t)][X_{lY}(t)] + [B(t)][X_{gY}(t)], \\
\frac{d[X_{lY}(t)]}{dt} &= -[X_{gY}(t)][X_{lY}(t)] + [B(t)][X_{lY}(t)], \\
\frac{d[B(t)]}{dt} &= -[B(t)][X_{lY}(t)] - [B(t)][X_{gY}(t)] + 2[X_{gY}(t)][X_{lY}(t)].
\end{aligned}$$

Assuming that the normalization step in the comparison module has already been performed, the species $(X_{gY}(t), X_{lY}(t), B(t))$ will converge to the state $(1, 0, 0)$ if $X_{gY}(0) > X_{lY}(0)$, and to $(0, 1, 0)$ if $X_{gY}(0) < X_{lY}(0)$. For a proof of this result, please see [23].

4.5 Loading Module

The loading module copies the concentration of A into B using the following network:

$$\begin{aligned}
A &\xrightarrow{1} A + B \\
B &\xrightarrow{1} \emptyset
\end{aligned} \tag{10}$$

In this network, the species A is a catalyst. Let $[A] = [A(0)]$. The dynamical system corresponding to the network (10) is given by

$$\frac{d[B]}{dt} = [A] - [B].$$

At steady state, we get $[B^{ss}] = [A]$.

4.6 Subtraction Module

We define a new species C, whose steady-state concentration will store the difference between the concentrations of two species A and B. In particular, the subtraction module will compute $[C^{ss}] = \max(0, A - B)$. The reaction network corresponding to this is given by:

$$\begin{aligned}
A &\xrightarrow{1} A + C \\
B &\xrightarrow{1} B + H \\
C &\xrightarrow{1} \emptyset \\
C + H &\xrightarrow{1} \emptyset
\end{aligned} \tag{11}$$

In this network, the species A and B are catalysts. Let $[A] = [A(0)]$ and $[B] = [B(0)]$. Then the dynamics corresponding to network (11) is given by

$$\begin{aligned}
\frac{d[C(t)]}{dt} &= [A] - [C(t)][H(t)] - [C(t)], \\
\frac{d[H(t)]}{dt} &= [B] - [C(t)][H(t)].
\end{aligned}$$

This implies that at steady state,

$$[C^{ss}] = \begin{cases} [A] - [B], & \text{if } [A] > [B], \\ 0, & \text{if } [A] \leq [B]. \end{cases}$$

5 Oscillation Module

In the oscillation module, we present a mechanism for time-scale separation using a clock derived from *Hopf dynamics* [13, 20]. The Hopf dynamics is turned into a discrete-time signal such that only one group of reactions is activated at a time. (see *Oscillator semantics* at the end of this section for a detailed explanation).

Note that it is possible to use a standard oscillation module as in [23]. However, such oscillations are not guaranteed to maintain constant amplitude and may attenuate over time, leading to errors that accumulate. Consequently, the reaction network implementation may fail to accurately match the implementation of SVMs. To obtain oscillations with constant amplitude, we instead employ a Hopf oscillator, and assume the existence of an external readout that converts this dynamics into a clock signal controlling the sequential execution of reactions (see Remark 5.1 for details).

The Hopf dynamical system [13, 20] is given by:

$$\begin{aligned} \frac{d[X(t)]}{dt} &= \mu[X(t)] - [X(t)]^3 - [Z(t)]^2[X(t)] - \omega[Z(t)], \\ \frac{d[Z(t)]}{dt} &= \mu[Z(t)] - [Z(t)]^3 - [X(t)]^2[Z(t)] + \omega[X(t)]. \end{aligned} \tag{12}$$

One can show that the trajectories of this dynamical system converge to a stable limit cycle of radius $\sqrt{\mu}$ and period $T = 2\pi/\omega$. We also remark that Equation (12) can be realized via mass-action kinetics using dual-rail encoding.

Given the oscillator dynamics, we obtain a discrete-time signal by computing a phase and then discretizing it as follows. We compute the current clock phase of the oscillator $\theta(t)$ by

$$\theta(t) = \text{atan2}([Z(t)], [X(t)]) \bmod 2\pi. \tag{13}$$

Here, the function $\text{atan2}(z, x)$ returns the angle between the x-axis and the point (x, z). Next, we discretize the phase into $n_{\texttt{clock}}$ slots as follows:

$$k(t) = \left\lfloor \frac{n_{\texttt{clock}}}{2\pi} \theta(t) \right\rfloor \bmod n_{\texttt{clock}}, \tag{14}$$

where $k(t)$ denotes the active oscillator slot index. We then define $O(t) \in \{0, 1\}^{n_{\texttt{clock}}}$ by

$$\text{For } i \in \{0, \ldots, n_{\texttt{clock}} - 1\}, \quad O_i(t) := \begin{cases} 1, & \text{if } i = k(t), \\ 0, & \text{otherwise.} \end{cases}$$

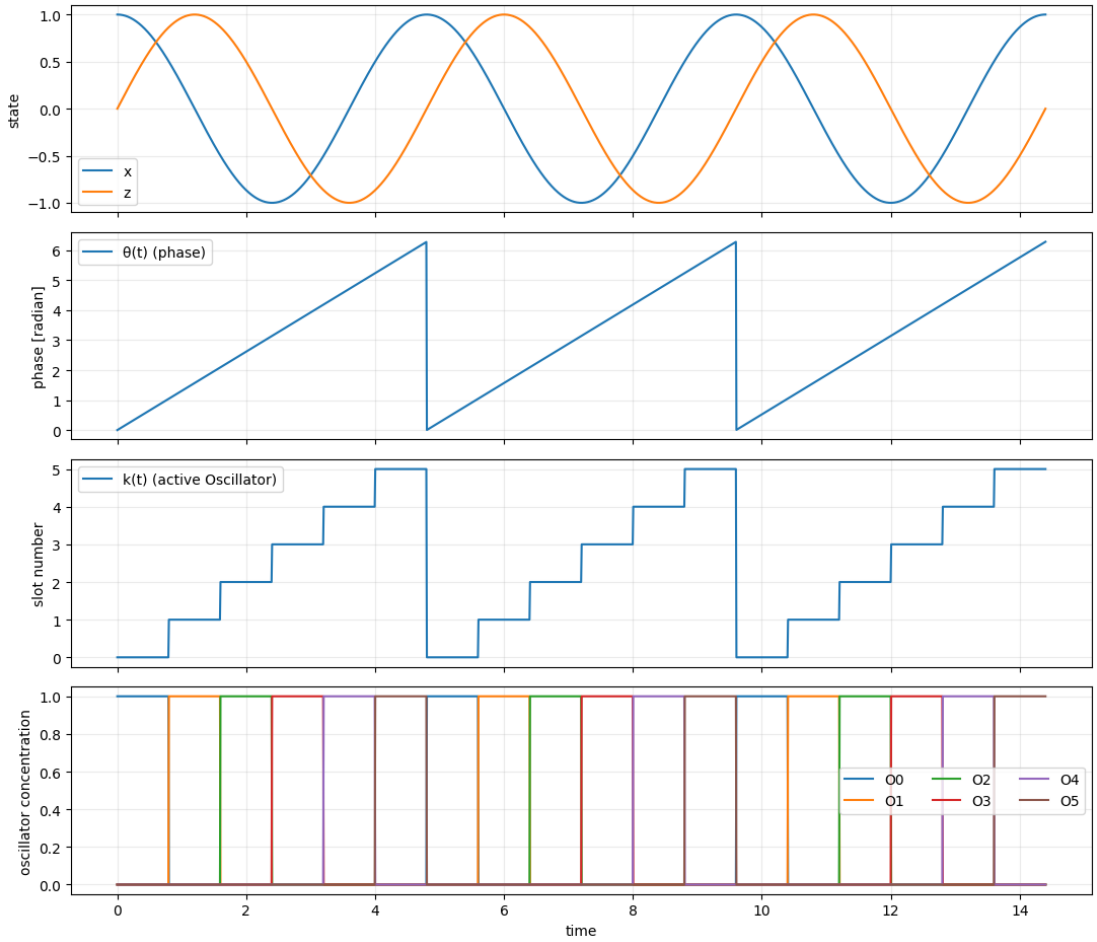


Figure 3: Hopf oscillator dynamics for $n_{\texttt{clock}} = 6$. The figure shows the time evolution of (i) $[X(t)]$ and $[Z(t)]$, (ii) the phase $\theta(t)$, (iii) the discrete index $k(t)$, and (iv) the indicator variables $O_i(t)$.

Remark 5.1. The phase computation and discretization in Equations (13) and (14) are not part of the reaction dynamics, but rather constitute a readout operation applied to the observables $x(t)$ and $z(t)$. We do not implement this step chemically; instead, we assume the existence of an external measurement device capable of computing this clock signal. This distinction between intrinsic dynamics and external readouts is standard in bioengineering applications.

Oscillator semantics: Every reaction in our SVM implementation is associated with an oscillator index $i \in \{0, \ldots, n_{\texttt{clock}} - 1\}$. If the intrinsic rate of a reaction r is $a_r(\cdot)$, then its effective propensity in the oscillator molecules is given by

$$a_r^{\texttt{eff}}(\cdot, t) = O_i(t)\, a_r(\cdot) \quad \text{with } i \in \{0, \ldots, n_{\texttt{clock}} - 1\}.$$

The oscillator species O_i acts as a binary gate for its associated reactions. At time t, when $O_i(t) = 1$, the corresponding reactions are active and proceed at rates, i.e., $a_r^{\texttt{eff}}(\cdot, t) = a_r(\cdot)$. When $O_i(t) = 0$, the effective rate is zero and the reactions are inactive. In this way, the oscillator enforces a sequential execution of reactions according to the clock signal. Moreover, the oscillator molecule functions as a catalyst: it modulates the reaction rates without being altered by the reactions.

6 Assignment Module

Inspired by [7], we develop the assignment module, which divides the input samples into batches and loads them into the input species set. This strategy allows us to parallelize our computation and allows faster execution of our program.

To formulate this, assume we have p samples, each of dimension d. In each iteration, we aim to feed $\tilde{p}$ samples (typically $\tilde{p}$ divides p) into the input layer. We divide the p samples across $\tilde{p}$ parallel input lanes. For every lane $l \in \{1, \ldots, \tilde{p}\}$, the eligible sample indices form the set

$$\mathcal{I}_l := \{l, \tilde{p} + l, 2\tilde{p} + l, \ldots\ p - \tilde{p} + l\}.$$

In every iteration, a sample is chosen from every lane. These samples are fed into the input layer. Over successive iterations, the active samples in each $\mathcal{I}_l$ are shifted through to be chosen. After the last sample is chosen, we cycle back to the first sample and continue the process.

For simplicity, we assume that all sample values are non-negative. We define

- The sample species set: $\{I_1^i, I_2^i, I_3^i\}$, where $1 \le i \le p$, represents the set of input samples.
- The input species set: $\{X_1^l, X_2^l, Y^l\}_{l=1}^{\tilde{p}}$, where $1 \le l \le \tilde{p}$, represents the set of input samples fed into the network at a time.
- The order species set: $\{C_i^l\}$, where $1 \le i \le p$ and $1 \le l \le \tilde{p}$.
- The auxiliary order species set: $\{\tilde{C}_i^l\}$, where $1 \le i \le p$ and $1 \le l \le \tilde{p}$.

These species support the assignment module, which operates in three phases. The reactions are

$$\begin{aligned}
O_0: &\quad C_i^l + I_q^i \xrightarrow{1} C_i^l + I_q^i + X_q^l \qquad X_q^l \xrightarrow{1} \emptyset \\
&\quad C_i^l + I_3^i \xrightarrow{1} C_i^l + I_3^i + Y^l \qquad Y^l \xrightarrow{1} \emptyset \\
O_1: &\quad C_i^l \xrightarrow{k} \tilde{C}_i^l \\
O_2: &\quad \tilde{C}_o^l \xrightarrow{k} C_{\tilde{p}+o}^l \qquad \tilde{C}_{p-\tilde{p}+l}^l \xrightarrow{k} C_l^l \qquad \tilde{C}_j^l \xrightarrow{k} C_j^l
\end{aligned}$$

where $q \in \{1, 2\}$, $o \in \mathcal{I}^{\text{“}}{}_l$, with $\mathcal{I}^{\text{“}}{}_l := \{l, \tilde{p} + l, 2\tilde{p} + l, \ldots, p - 2\tilde{p} + l\}$, and $j \notin \mathcal{I}_l$.

To initialize the system, we set the initial concentrations of the order species $\{C_i^l\}$ as

$$[C_i^l(0)] = \begin{cases} 1, & \text{if } l = i, \\ 0, & \text{otherwise,} \end{cases} \quad \text{for every } 1 \le i \le p \text{ and } 1 \le l \le \tilde{p}.$$

This means that for each lane index $l \in \{1, \ldots, \tilde{p}\}$, only C_l^l are active initially, while all other order species remain inactive. For the auxiliary order species $\{\tilde{C}_i^l\}$, all concentrations are initialized to zero as follows:

$$[\tilde{C}_i^l(0)] = 0 \quad \text{for every } 1 \le i \le p \text{ and } 1 \le l \le \tilde{p}.$$

The concentrations of the input species $\{Y^l\}$ and $\{X_q^l\}$ (for $q \in \{1, 2\}$ and $l \in \{1, \ldots, \tilde{p}\}$) are initialized to arbitrary nonnegative values:

$$[Y^l(0)] \cup [X_q^l(0)] \in \mathbb{R}_{\ge 0} \quad \text{for every } 1 \le q \le 3 \text{ and } 1 \le l \le \tilde{p}.$$

In Reaction Phase O_0, the assignment operation is performed. Specifically, the reactions:

$$\begin{aligned} C_i^l + I_q^i &\xrightarrow{1} C_i^l + I_q^i + X_q^l \\ C_i^l + I_3^i &\xrightarrow{1} C_i^l + I_3^i + Y^l \\ X_q^l &\xrightarrow{1} \emptyset \\ Y^l &\xrightarrow{1} \emptyset \end{aligned}$$

This reaction network collectively stores the sum of the term-wise product of the order species concentration $[C_i^l]$ with the sample input species I_1^i, I_2^i and I_3^i into the corresponding X_1^l, X_2^l and Y^l. Since the catalyst concentrations satisfy $[C_i^l] = 0$ for all $l \neq i$ and $[C_l^l] = 1$, the system effectively selects and loads only one active sample per lane during each iteration. Thus, $\tilde{p}$ samples are loaded into the input layer parallel.

In the Phase O_1, the active order species are copied into the auxiliary species via the reaction:

$$C_i^l \xrightarrow{k} \tilde{C}_i^l$$

This mechanism preserves the current assignment state in preparation for the rotation of order species in the next iteration.

In Reaction Phase O_2, the system initiates a cyclic shift of the active samples across lanes. The reactions:

$$\tilde{C}_o^l \xrightarrow{k} C_{\tilde{p}+o}^l \qquad \tilde{C}_{p-\tilde{p}+l}^l \xrightarrow{k} C_l^l \qquad \tilde{C}_j^l \xrightarrow{k} C_j^l$$

perform the following: the active block in each position is shifted forward by $\tilde{p}$, going to the next element in their respective $\mathcal{I}_l$, except for the last element of the lane, which is cyclically wrapped around to the first position. Any remaining auxiliary species that do not participate in the cycle are transferred to their corresponding order species.

Remark 6.1. As per the above convention, reactions are formed for all pairs of (l, i). However, for a given l, only those $i \in \mathcal{I}_l$ have C_i^l as active at some time during the network. Consequently, reactions involving those inactive indices will never contribute to the dynamics. In our Python implementation, we iterate only over those indices that can be active and skip inactive indices to reduce computational overhead.

Together, these phases constitute an automated mechanism for batch-wise data assignment, ensuring that $\tilde{p}$ samples are loaded in parallel at each iteration and that the active sample in every lane evolves cyclically without external logic.

7 Encoding the Feedforward Module of SVM

In the following sections, we describe the computation done for a single sample passing through the reaction network. We omit the lane indices used previously. The weights and biases are shared across lanes. But the input, and intermediate species are lane specific, unless stated otherwise.

Given an input sample $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ with an associated label $y \in \{-1, 1\}$, a weight vector $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$, and a bias term $b \in \mathbb{R}$, our objective is to determine whether the classification is correct by evaluating $y(\mathbf{w} \cdot \boldsymbol{x} + b)$ against 1.

In this section, we construct reaction networks that compute the products $w_i x_i$, sum them together with the bias b, and then multiply the resulting sum by y (Steps 1–3), allowing the resulting value to be compared with 1 (Steps 4–5). The complete computation is performed through a sequence of five steps, utilizing the oscillatory molecules O_3, O_4, O_5, O_6, O_7 and O_8.

In Sections 7 and 8, we denote by W_i the species representing the weight w_i, by X_i the species representing x_i, by Y the species representing y, and by B representing the bias b.

Step 1: We introduce the species WX_i, whose steady state concentration will equal the product of W_i and X_i. Specifically, we use the multiplication module (7) and consider the following network:

$$\begin{aligned} W_1 + X_1 + O_3 &\xrightarrow{1} W_1 + X_1 + WX_1 + O_3 \\ WX_1 + O_3 &\xrightarrow{1} O_3 \\ &\vdots \\ W_d + X_d + O_3 &\xrightarrow{1} W_d + X_d + WX_d + O_3 \\ WX_d + O_3 &\xrightarrow{1} O_3 \end{aligned} \tag{15}$$

In this network, the species W_i, X_i are catalysts. Let $[W_i] = [W_i(0)]$ and $[X_i] =$

$[X_i(0)]$. The dynamical system corresponding to the network (15) is given by

$$\frac{d[WX_i(t)]}{dt} = [W_i][X_i][O_3] - [WX_i(t)][O_3] \text{ for every } 1 \leq i \leq d.$$

At steady state, we get $[WX_i{}^{ss}] = [W_i][X_i]$.

Step 2: We introduce a new species P, whose steady-state concentration represents the sum of all WX_i plus B. In particular, we use the addition module (6) and consider the following network:

$$\begin{aligned} WX_1 + O_4 &\xrightarrow{1} WX_1 + P + O_4 \\ &\vdots \\ WX_d + O_4 &\xrightarrow{1} WX_d + P + O_4 \\ B + O_4 &\xrightarrow{1} B + P + O_4 \\ P + O_4 &\xrightarrow{1} O_4 \end{aligned} \tag{16}$$

In this network, the species WX_i, B are catalysts. Let $[WX_i] = [WX_i(0)]$ and $[B] = [B(0)]$. The dynamical system corresponding to the network (16) is given by

$$\frac{d[P(t)]}{dt} = \sum_{i=1}^{d} [WX_i][O_4] + [B][O_4] - [P(t)][O_4].$$

At steady state, we get $[P^{ss}] = \sum_{i=1}^{d} [WX_i] + [B]$.

Step 3: We introduce the species Q, whose steady-state concentration corresponds to the product of P and Y. Specifically, we use the multiplication module (7) and consider the following network:

$$\begin{aligned} P + Y + O_5 &\xrightarrow{1} P + Y + Q + O_5 \\ Q + O_5 &\xrightarrow{1} O_5 \end{aligned} \tag{17}$$

In this network, the species P, Y are catalysts. Let $[P] = [P(0)]$ and $[Y] = [Y(0)]$. The dynamical system corresponding to network (17) is

$$\frac{d[Q(t)]}{dt} = [P][Y][O_5] - [Q(t)][O_5].$$

At steady state, we get $[Q^{ss}] = [P][Y]$. Combining this with the results from Steps 1 and 2, it follows that

$$[Q^{ss}] = \Big(\sum_{i=1}^{N} [WX_i] + [B] \Big) [Y].$$

Next, we compare the value stored in Q with 1. Depending on whether the associated label is correct (i.e., whether the concentration of Q is greater or less than 1), the concentration of one of the two flag species (K_{lQ} and K_{gQ}) converges to 1, while the concentration of the other species converges to 0.

We divide this process into two sequential steps: Step 4, which normalizes the flag species, and Step 5, which performs the comparison. To ensure these steps occur sequentially, we employ oscillatory molecules. Oscillation molecule O_7 is used in Step 4, while O_8 is used in Step 5. This guarantees that the reactions in Step 4 complete before those in Step 5 begin.

We employ dual-rail encoding to ensure that all species concentrations remain non-negative. Since the quantity Q can take negative values, we compute it using the subtraction module (Equation (11)), which produces an effective concentration proportional to $\max(Q^+ - Q^-, 0)$. This introduces a rectification that clamps negative values of Q to zero. However, this does not affect correctness, as the subsequent comparison only depends on whether Q exceeds the threshold; if Q were negative, clamping it to zero preserves the outcome of the comparison. We utilise O_6 molecule in this step.

Step 4: We introduce a catalytic species K along with the flag species K_{lQ} and K_{gQ}. We set the concentration of K to 1, i.e., $[K] = [K(0)] = 1$, and designate K_{lQ} and K_{gQ} as flag species, with the sum of their initial concentrations equal to 1. Using the comparison module (8), the network implementing this comparison is given by

$$\begin{aligned} Q + K_{gQ} + O_7 &\xrightarrow{1} K_{lQ} + Q + O_7 \\ K_{lQ} + K + O_7 &\xrightarrow{1} K_{gQ} + K + O_7 \end{aligned} \tag{18}$$

In this network, the species Q is a catalyst. Let $[Q] = [Q(0)]$ and let $[K_{lQ}(0)] + [K_{gQ}(0)] = 1$. The dynamical system corresponding to network (18) is given by

$$\begin{aligned} \frac{d[K_{lQ}(t)]}{dt} &= [Q][K_{gQ}(t)][O_7] - [K_{lQ}(t)][K][O_7], \\ \frac{d[K_{gQ}(t)]}{dt} &= -[Q][K_{gQ}(t)][O_7] + [K_{lQ}(t)][K][O_7]. \end{aligned}$$

Adding the two equations above gives us that $[K_{lQ}(t)] + [K_{gQ}(t)]$ is preserved for all time $t \geq 0$. Consequently, the steady-state concentrations of K_{lQ} and K_{gQ} will be set as follows:

$$[K_{lQ}^{ss}] = \frac{[Q]}{[Q] + [K]} = \frac{[Q]}{[Q] + 1}, \qquad [K_{gQ}^{ss}] = \frac{[K]}{[Q] + [K]} = \frac{1}{[Q] + 1}.$$

Step 5: We compare Q and K by using the approximate majority algorithm (see Section 4.4). Specifically, the flag species K_{lQ} and K_{gQ} are set such that, if $[Q] > [K] = 1$, K_{lQ} converges to 1 and K_{gQ} converges to 0; conversely, if $[Q] < [K] = 1$, K_{lQ} converges to 0 and K_{gQ} converges to 1. Using the approximate majority module (9),

the network that implements this behavior is given by

$$\begin{aligned} K_{gQ} + K_{lQ} + O_8 &\xrightarrow{1} K_{lQ} + U + O_8 \\ U + K_{lQ} + O_8 &\xrightarrow{1} 2K_{lQ} + O_8 \\ K_{lQ} + K_{gQ} + O_8 &\xrightarrow{1} K_{gQ} + U + O_8 \\ U + K_{gQ} + O_8 &\xrightarrow{1} 2K_{gQ} + O_8 \end{aligned} \tag{19}$$

The dynamical system corresponding to network (19) is given by

$$\begin{aligned} \frac{d[K_{lQ}(t)]}{dt} &= -[K_{gQ}(t)][K_{lQ}(t)][O_8] + [U(t)][K_{lQ}(t)][O_8], \\ \frac{d[K_{gQ}(t)]}{dt} &= -[K_{gQ}(t)][K_{lQ}(t)][O_8] + [U(t)][K_{gQ}(t)][O_8], \\ \frac{d[U(t)]}{dt} &= [K_{gQ}(t)][K_{lQ}(t)][O_8] - [U(t)][K_{gQ}(t)][O_8] + [K_{lQ}(t)][K_{gQ}(t)][O_8] \\ &\quad - [U(t)][K_{gQ}(t)][O_8]. \end{aligned}$$

The analysis of this dynamical system follows the discussion in Section 4.4. In conclusion, if $K_{lQ}(t)$ converges to 1 and $K_{gQ}(t)$ converges to 0, this indicates that $[Q] > [K]$ and the associated label is correct. Conversely, if $K_{lQ}(t)$ converges to 0 and $K_{gQ}(t)$ converges to 1, this indicates that $[Q] < [K]$ and the associated label is incorrect.

8 Backpropagation Module of SVM using Gradient Descent

In this section, we construct reaction networks corresponding to the backpropagation component of the SVM, that is, the adjustment of the weight vector $\mathbf{w}$ and the bias b via gradient descent.

Recall from Section 7 that we determined whether a given input sample $\boldsymbol{x}$ with label y is correctly classified under $\mathbf{w}$ and b. Given a regularization parameter λ and a learning rate $\eta > 0$, the reaction networks implemented here realize Equations (3) and (4) from the iterative training algorithm, given by:

- Misclassification case ($y(\mathbf{w} \cdot \boldsymbol{x} + b) < 1$):

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta(\lambda \mathbf{w} - y\boldsymbol{x}) \\ b &\leftarrow b + \eta y \end{aligned}$$

- No misclassification case ($y(\mathbf{w} \cdot \boldsymbol{x} + b) \geq 1$):

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta\lambda \mathbf{w} \\ b &\leftarrow b \end{aligned}$$

We first compute the scaled weight $(1-\eta\lambda)\mathbf{w}$ (Step 1), since this term appears in both cases regardless of classification outcome. Next, we compute the products ηy and $\eta y\boldsymbol{x}$, which are required only in the misclassification case (Step 2). Finally, we apply the updates to the bias and the weight vector (Step 3). The complete computation is performed in two phases, utilizing the oscillatory molecules O_9 and O_{10}.

Step 1: We introduce the species R_i, whose steady state concentration will equal the product of W_i and $\alpha = 1-\eta\lambda$ (update for the weight). Specifically, we we use the multiplication module (7) and consider the following network:

$$\begin{aligned} O_9 + \alpha + W_1 &\to O_9 + \eta + \lambda\alpha + W_1 + R_1 \\ R_1 + O_9 &\to O_9 \\ &\vdots \\ O_9 + \alpha + W_d &\to O_9 + \eta + \lambda\alpha + W_d + R_d \\ R_d + O_9 &\to O_9 \end{aligned} \tag{20}$$

In this network, the species W_i is a catalyst. Let $[W_i] = [W_i(0)]$. The dynamical system corresponding to the network (20) is given by

$$\frac{d[R_i(t)]}{dt} = \alpha[W_i][O_9] - [R_i(t)][O_9].$$

At steady state, we get $[R_i^{ss}] = \alpha[W_i] = (1-\eta\lambda)[W_i]$.

Next, we load the bias. We introduce the species Z, whose steady-state concentration B. Using the loading module (10), the reaction network implementing this operation is given by:

$$\begin{aligned} B + O_9 &\to O_9 + B + Z \\ Z + O_9 &\to O_9 \end{aligned}$$

In this network, the species B is catalyst. Let $[B] = [B(0)]$. The dynamical system corresponding to this network is

$$\frac{d[Z(t)]}{dt} = [B][O_9] - [Z(t)][O_9].$$

At steady state, we get $[Z^{ss}] = [B]$.

Note that both the species, R_i and Z are common to all the lanes.

Step 2: We introduce two species XY_i and Sb, whose steady-state concentrations represent the quantities $\eta K_{gQ} X_i Y$ (update for the weight) and $\eta K_{gQ} Y$ (update for the bias), respectively. In particular, we use the multiplication module (7) and consider the

following network:

$$\begin{aligned}
O_9 + \eta + Y + X_1 + K_{gQ} &\to O_9 + \eta + Y + X_1 + XY_1 + K_{gQ} \\
XY_1 + O_9 &\to O_9 \\
&\vdots \\
O_9 + \eta + Y + X_d + K_{gQ} &\to O_9 + \eta + Y + X_d + XY_d + K_{gQ} \\
XY_d + O_9 &\to O_9 \\
O_9 + \eta + Y + K_{gQ} &\to O_9 + \eta + Y + Sb + K_{gQ} \\
Sb + O_9 &\to O_9
\end{aligned}$$

In this network, the species X_i, Y, K_{gQ} are catalysts. Let $[X_i] = [X_i(0)]$, $[Y] = [Y(0)]$, and $[K_{gQ}] = K_{gQ}(0)$. The dynamical system corresponding to this network is

$$\begin{aligned}
\frac{d[XY_i(t)]}{dt} &= \eta[K_{gQ}(t)][X_i][Y][O_9] - [XY_i(t)][O_9], \\
\frac{d[Sb(t)]}{dt} &= \eta[K_{gQ}(t)][Y][O_9] - [Sb(t)][O_9].
\end{aligned}$$

At steady state, we get $[XY_i^{ss}] = \eta[K_{gQ}][X_i][Y]$ and $[Sb^{ss}] = \eta[K_{gQ}][Y]$. Note that the catalyst species K_{gQ} acts as a switch: it takes the value $[K_{gQ}] = 1$ in the misclassification case and $[K_{gQ}] = 0$ in the no-misclassification case, thereby enabling or preventing weight updates.

Note that the species XY_i and Sb are shared across lanes, but each lane contributes to their production independently. In particular, each lane computes its local update term, and these contributions are accumulated into the shared species through reactions whose rate constants are scaled by $1/\tilde{p}$. At steady state, the accumulated quantities correspond to the mean over the $\tilde{p}$ samples processed in parallel. Thus,

$$[XY_i^{ss}] = \tfrac{1}{\tilde{p}} \textstyle\sum_{l=1}^{\tilde{p}} \eta[K_{gQ}^{(l)}][X_i^{(l)}][Y^{(l)}], \quad [Sb^{ss}] = \tfrac{1}{\tilde{p}} \textstyle\sum_{l=1}^{\tilde{p}} \eta[K_{gQ}^{(l)}][Y^{(l)}].$$

Step 3: We now update the weight and the bias.

First, we recall from Step 1 that the species R_i directly encodes the scaled weight $(1 - \eta\lambda)W_i$. We directly use R_i as the base weight and add the correction term XY_i corresponding to the hinge-loss update. We directly combine R_i and the correction term XY_i to form the updated weight. To update the bias, we store the sum of Z and Sb into B The reaction network implementing this operation is given by:

$$\begin{aligned}
R_i + O_{10} &\to R_i + O_{10} + W_i \\
XY_i + O_{10} &\to XY_i + O_{10} + W_i \\
Z + O_{10} &\to Z + O_{10} + B \\
Sb + O_{10} &\to Sb + O_{10} + B \\
W_i + O_{10} &\to O_{10} \\
B + O_{10} &\to O_{10}
\end{aligned} \tag{21}$$

In this network, the species R_i, XY_i, Sb and Z are catalysts. Let $[R_i] = [R_i(0)]$, $[XY_i] = [XY_i(0)]$, $[Sb] = [Sb(0)]$ and $[Z] = [Z(0)]$. The dynamical system corresponding to this network is

$$\frac{d[W_i(t)]}{dt} = [R_i][O_{10}] + [XY_i][O_{10}] - [W_i(t)][O_{10}],$$
$$\frac{d[B(t)]}{dt} = [Z][O_{10}] + [Sb][O_{10}] - [B(t)][O_{10}].$$

At steady state, we obtain $[W_i^{ss}] = [R_i] + [XY_i]$ and $[B^{ss}] = [Z] + [Sb]$. As a result, Equations (3) and (4) from the iterative training algorithm are implemented (Equation (5) in our parallel batch setting). After the update is applied, all intermediate species are reset in O_{11} by using decay reactions, restoring the system to its initial state for the next iteration. This process is repeated across all batches until a predefined number of iterations is reached.

9 Results and Analysis

In this section, we compare the actual and predicted values of the learned SVM parameters across the reaction network.

9.1 Experimental Setup

To validate the functionality of our reaction network, we perform simulations using a CRN-based SVM and compare its performance against a standard soft-margin SVM baseline, implemented in Python. We generate a two-dimensional linearly separable dataset using a random generator and introduce controlled noise. Each sample is assigned a binary label in $\{+1, -1\}$.

In our experimental setup, we consider $p = 10$ training samples (evenly divided between the two classes) and 4 test samples. We use a minibatch size of $\tilde{p} = 2$ and train the model for 100 epochs.

9.2 Feedforward Network and Learning Module Simulation Results

We compare the actual and predicted values of the learned SVM parameters. The table below presents a comparison of the true and computed values for weights and bias, along with their respective errors.

The results indicate that the average error in predicting the parameters is approximately 0.033%, demonstrating that the CRN-based approach closely approximates the standard SVM training process.

To further validate these results, we visualize the evolution of the learned weights and bias throughout the training process. Figure 4 provides a consolidated view of the

Parameter	Actual Value	Net Predicted Value	Absolute Error
w_1	1.2548	1.2548	0.0
w_2	-0.2598	-0.2598	0.0
b	-0.1	-0.0999	0.0001
Average Absolute Error			**0.0001**

Table 1: Comparison of Actual and Predicted SVM Weights

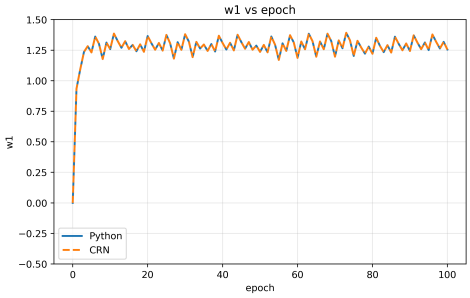
dynamics of the weight parameters, the bias term, and the comparison between the predicted and actual SVM hyperplanes. The plots illustrate how the model's predicted values align with the actual values over the course of 100 training epochs.

- Figure 4a and Figure 4b (Weight 1 and Weight 2): Initially, the weights exhibit a sharp incline and decline respectively, followed by stabilization after approximately 20 epochs.This trend reflects the gradient-based optimization process, where rapid updates occur in the early stages, and convergence is reached as the learning progresses. The predicted values (orange dashed lines) closely track the actual values (solid blue lines), indicating strong predictive accuracy.
- Figure 4c (Bias): The bias shows a similar pattern of rapid early changes before stabilizing. While slight deviations between the predicted and actual bias exist, the overall trend suggests effective approximation of the learning dynamics.
- Figure 4d and Figure 4e (Hyperplane Comparison): The actual hyperplane from SVM (green solid line) is compared with the predicted hyperplane (black dashed line). In train as well as test case, the hyperplane is able to divide the samples into their respective classes. The CRN hyperplane also closely follows the traditional hyperplane. The overall similarity suggests that the model effectively captures the underlying decision boundary.

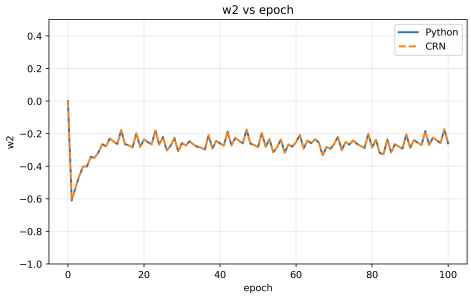
The close alignment between predicted and actual values demonstrates the reliability of the CRN-based predictive model, while also suggesting potential areas for further refinement to enhance precision.

10 Discussion

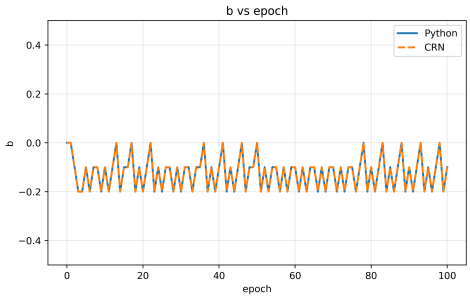
We have presented a reaction network–based framework for implementing soft-margin support vector machines (SVMs). The proposed method realizes both the feedforward and backpropagation modules of the SVM through a collection of submodules. Specifically, the design incorporates addition, subtraction, multiplication, comparison, and



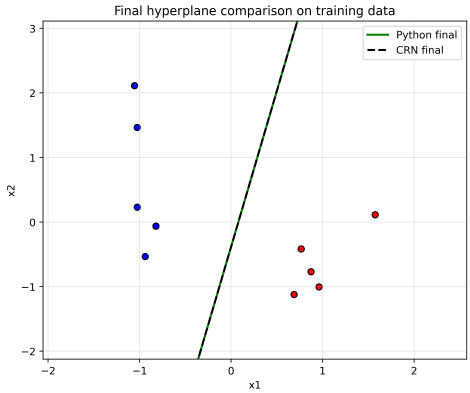
(a) Weight 1 as a function of epochs



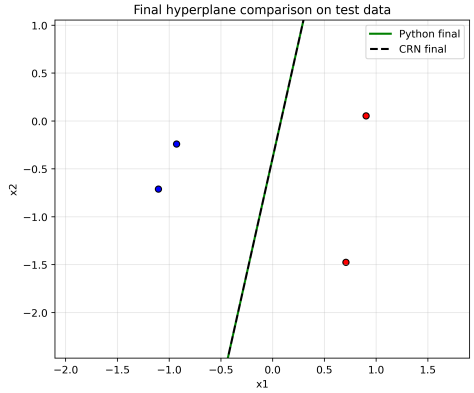
(b) Weight 2 as a function of epochs



(c) Bias as a function of epochs



(d) Hyperplane on Train Data



(e) Hyperplane on Test Data

Figure 4: Representation of Parameter Evolution and Hyperplane

approximate majority modules. Sequential execution of reactions is achieved via time-scale separation, and implemented using an Hopf oscillator. To handle the possibility of negative weights and biases, we utilize dual-rail encoding.

Our work intersects with several other works that have used reaction networks to implement machine learning algorithms. Related approaches include the modeling of

neural networks through chemical reaction networks [2], the implementation of Boltzmann machines [15, 16], and the computation of maximum likelihood estimates [9].

The present study opens multiple avenues for further investigation. One direction is to rigorously analyze the computational complexity of our implementation, in terms of the number of molecular species, the number of reactions, and the convergence rates of the individual computation steps, as in [3]. Another direction involves the computational optimality of our design in the sense of [2]. More specifically, if the reaction network is represented by the dynamical system

$$\dot{x}(t) = f(y, x(t)) \quad \text{with} \quad x(t) \in \mathbb{R}^n_{\geq 0},$$

two key questions arise:

1. *Exponential reliability:* Does there exist a function $\zeta : \mathbb{R}^p_{>0} \to \mathbb{R}_{>0}$ such that

   $$|x(t) - \xi(y)| \leq |x(0) - \xi(y)|e^{-\zeta(y)t},$$

   where $\lim_{t\to\infty} x(t) = \xi(y)$?

2. *Finite-time convergence to compact set:* Do the trajectories converge to a compact set containing the equilibrium in finite time? That is, does there exist a compact set $K \subset \mathbb{R}^n_{\geq 0}$ and a function $T : \mathbb{R}^p_{>0} \to \mathbb{R}_{>0}$ such that $x(t) \in K$ for all $t \geq T(y)$ and $x(0) \in \mathbb{R}^n_{\geq 0}$?

Finally, it is natural to explore extensions of the proposed framework. Possible directions include adapting the scheme to incorporate the *kernel trick* to enable the classification of nonlinear data.

11 Data accessibility statement

All relevant codes can be found here: https://github.com/AmeyChoudhary/svm-implementation-using-crn/tree/main

References

[1] L. Adleman, *Molecular computation of solutions to combinatorial problems*, Science (1994), 1021–1024.

[2] D. Anderson, B. Joshi, and A. Deshpande, *On reaction network implementations of neural networks*, J. R. Soc. Interface **18** (2021), no. 177, 20210031.

[3] D. F. Anderson and B. Joshi, *Chemical mass-action systems as analog computers: Implementing arithmetic computations at specified speed*, Theor. Comput. Sci. **1025** (2025), 114983.

[4] L. Cardelli and A. Csikász-Nagy, *The cell cycle switch computes approximate majority*, Scientific reports **2** (2012), no. 1, 656.

[5] K. Cherry and L. Qian, *Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks*, Nature **559** (2018), no. 7714, 370–376.

[6] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[7] Y. Fan, X. Zhang, C. Gao, and D. Dochain, *Automatic implementation of neural networks through reaction networks–part i: Circuit design and convergence analysis*, arXiv preprint arXiv:2311.18313 (2023).

[8] Martin Feinberg, *Foundations of chemical reaction network theory*, Applied Mathematical Sciences, vol. 202, Springer, Cham, 2019.

[9] M. Gopalkrishnan, *A scheme for molecular computation of maximum likelihood estimators for log-linear models*, International Conference on DNA-Based Computers, Springer, 2016, pp. 3–18.

[10] C. Guldberg and P. Waage, *Studies Concerning Affinity*, CM Forhandlinger: Videnskabs-Selskabet I Christiana **35** (1864), no. 1864, 1864.

[11] I. Guyon, B. Boser, and V. Vapnik, *Automatic capacity tuning of very large vc-dimension classifiers*, Advances in neural information processing systems **5** (1992).

[12] S. Kang and J. Kim, *Noise-robust chemical reaction networks training artificial neural networks*, arXiv preprint arXiv:2410.11919 (2024).

[13] Yuri A. Kuznetsov, *Elements of applied bifurcation theory*, 3 ed., Springer, 2004.

[14] R. Pei, E. Matamoros, M. Liu, D. Stefanovic, and M. Stojanovic, *Training a molecular automaton to play a game*, Nat. Nanotechnol. **5** (2010), no. 11, 773.

[15] W. Poole, A. Ortiz-Munoz, A. Behera, N. Jones, T.E. Ouldridge, E. Winfree, and M. Gopalkrishnan, *Chemical boltzmann machines*, International Conference on DNA-Based Computers, Springer, 2017, pp. 210–231.

[16] W. Poole, T. Ouldridge, and M. Gopalkrishnan, *Autonomous learning of generative models with chemical reaction network ensembles*, J. R. Soc. Interface **22** (2025), no. 222, 20240373.

[17] W. Poole, T. Ouldridge, M. Gopalkrishnan, and E. Winfree, *Detailed balanced chemical reaction networks as generalized boltzmann machines*, arXiv preprint arXiv:2205.06313 (2022).

[18] L. Qian, E. Winfree, and J. Bruck, *Neural network computation with dna strand displacement cascades*, Nature **475** (2011), no. 7356, 368–372.

[19] F. Simmel, B. Yurke, and H. Singh, *Principles and applications of nucleic acid strand displacement reactions*, Chem. Rev. **119** (2019), no. 10, 6326–6369.

[20] Steven H. Strogatz, *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*, Westview Press, 2001.

[21] V. Vapnik, *The support vector method*, International conference on artificial neural networks, Springer, 1997, pp. 261–271.

[22] M. Vasic, C. Chalk, S. Khurshid, and D. Soloveichik, *Deep molecular programming: a natural implementation of binary-weight relu neural networks*, International Conference on Machine Learning, PMLR, 2020, pp. 9701–9711.

[23] M. Vasić, D. Soloveichik, and S. Khurshid, *Crn++: Molecular programming language*, Nat. Comput. **19** (2020), no. 2, 391–407.

[24] V. Virinchi, A. Behera, and M. Gopalkrishnan, *A stochastic molecular scheme for an artificial cell to infer its environment from partial observations*, International Conference on DNA-Based Computers, Springer, 2017, pp. 82–97.

[25] ———, *A reaction network scheme which implements the EM algorithm*, International Conference on DNA Computing and Molecular Programming, Springer, 2018, pp. 189–207.