Neural CRNs
A Natural Implementation of Learning in Chemical Reaction Networks

Rajiv Nagipogu

Advisor: Dr. John H. Reif
Organization

- DNA Computing & Molecular Computing
- Learning and Advantages of Molecular scale learning
- Literature Review
- ODEs of Neural ODEs
- Constructing Neural CRNs
  - Preliminaries
  - Forward phase CRNs
  - Backpropagation phase CRNs
  - Weight update CRNs
- Training and Classification by Neural CRNs
- Comparison of Neural CRNs w/ parent Neural ODE
- Extensions
  1. Nonlinear separation boundary
  2. ReLU activation networks
  3. Multi-layered networks
Molecular Computing & DNA Computing

**Molecular Computing**

Unconventional form of Computing

**DNA Computing**

Highly programmable

Cheaply available

Biocompatible

---

A CRN that implements $y = a + b$

Computing using chemical reaction networks

---

**Engineering: Using ‘nature’s toolbox,’ a DNA computer solves a complex problem**

BY Matthew Blakeslee  ·  MARCH 19, 2003

A DNA-based computer has solved a logic problem that no person could complete by hand, setting a new milestone for this infant technology that could someday surpass the electronic. The results were published on March 4 and will then run it.

**Doesn’t scale!**

Perform computation where silicon-based hardware can’t go!

Initially thought to solve NP-Hard problems owing to inherent parallelism

---


---


---

DNA as substrate for universal chemical kinetics

DNA based AND Circuit

DNA Data storage*
Neural Networks & Learning at Molecular scale

**Neural Networks**

- Powerful function approximators
- Can **learn** from data alone without any programming
- Trained network can generalize to unseen data

**Molecular-scale learning**

- Smart Therapeutics
- Targeted drug delivery
- Sensing and actuation at cellular level
- Long-term Monitoring of a cell
- Applications in Syn. Bio, Biotech

**Many applications!**

- Can perform tasks autonomously without explicitly programming the instructions!
- Can perform tasks in uncertain chemical environments! e.g., inside biological systems
- Some examples of chemical learning later!

Silicon based learning: Artificial Neural Networks & Backpropagation

\[ p_{\text{new}} = p_{\text{old}} - \eta \frac{\partial L}{\partial p} \]
How about *learning* in the chemical medium?

Prior Work
Prior Work: Chemical Implementation of Neural Networks (Hjelmfelt et al. 1991)

\[
I_{li}^* + C_i \rightleftharpoons X_{li} + C_i \\
X_{li} + B_i \rightleftharpoons X_{li}^* + A_i \\
X_{3i} + A_i \rightleftharpoons X_{3i}^* + B_i \\
X_{3i}^* \rightleftharpoons I_{li}^*
\]

CRN Implements the McCulloch-Pitts neuron

Spike observed when input is greater than a threshold
Prior Work: Online learning in a Chemical Perceptron (Banda et al., 2013)

**Mathematical Perceptron**
\[
\hat{y} = w_1 x_1 + w_2 x_2
\]

\[
\hat{y} = w_1 x_1 + w_2 x_2
\]

**Backpropagation Stage**
\[
w_i(t + 1) = w_i(t) + \alpha(y - \hat{y})x_i
\]

\[
w_i(t + 1) = w_i(t) + \alpha(y - \hat{y})x_i
\]

**Forward Stage**

\[
X_i \xrightarrow{W_i} Y + X_i Y,
\]

\[
X_i + Y \rightarrow \lambda.
\]

\[
X_i \xrightarrow{W_i} Y + X_i Y,
\]

\[
X_i + Y \rightarrow \lambda.
\]

\[
[Y]_{ss} = \sum_i [W_i]_0 [X_i]_0
\]

\[
[Y]_{ss} = \sum_i [W_i]_0 [X_i]_0
\]

**CRN-based chemical perceptron**

\[
Y + \hat{Y} \rightarrow \lambda \quad \rightarrow \quad \hat{Y} \rightarrow W^\Theta
\]

\[
Y + \hat{Y} \rightarrow \lambda \quad \rightarrow \quad \hat{Y} \rightarrow W^\Theta
\]

\[
Y \xrightarrow{s_i} W^\Theta
\]

\[
Y \xrightarrow{s_i} W^\Theta
\]

\[
W^\Theta \xrightarrow{x_i Y} W_i,
\]

\[
W^\Theta \xrightarrow{x_i Y} W_i
\]

\[
W_i^\Theta + W_i \rightarrow \lambda
\]

\[
W_i^\Theta + W_i \rightarrow \lambda
\]

Calculates \(y - \hat{y}\)

Calculates approx. value of \(\alpha(y - \hat{y})x_i\)

\(W^\Theta\) or \(W^\Theta\) adjusts the weight concentration \(W_i\)

\(W^\Theta\) or \(W^\Theta\) adjusts the weight concentration \(W_i\)
Prior Work: Supervised learning using DNA circuits (Lakin et al., 2016)

Buffered DNA Strand displacement gates

Can be activated programmatically => Adaptive!

Buffered DSD circuit for: \( B + X \rightarrow X + Y \)

\[
B + x \rightarrow x + y \\
B' + x \rightarrow \Phi \\
\equiv
\]

\[
y_{ss} = wx_0 \\
w = \frac{[B]}{[B']} \\
B \\
x \rightarrow w \rightarrow y \\
B'
\]

An Amplifier repurposed as a multiplier circuit

ACS Synth. Biol. 2016, 5, 8, 885–897
Prior Work: Supervised learning using DNA circuits (Lakin et al., 2016)

\[
B + x \rightarrow x + y \\
B' + x \rightarrow \Phi
\]

\[
y = w_1 x_1 + w_2 x_2
\]

\[
y_{ss} = wx_0 \\
w = \frac{[B]}{[B']}
\]

\[
w_i(t + 1) = w_i(t) + \alpha(y - \hat{y})x_i
\]
Implementing digital computations using CRNs is nonideal, and could lead to **cumbersome reaction networks**. First-order approximations of gradients, leads to bad approximations.
Are Dynamical Systems Worth Looking at?

Lotka-Volterra Model

\[
\frac{dx}{dt} = \alpha x - \beta xy \\
\frac{dy}{dt} = \delta \beta xy - \gamma y
\]

Oscillatory predator-prey dynamics

Turns out...

\[
\alpha X + \beta Y \rightarrow \gamma Z \\
\frac{d[Z]}{dt} = [X]^\alpha [Y]^\beta
\]

CRNs are dynamical systems!

https://strimas.com/post/lotka-volterra/
Neural Network-like Learning using Dynamical Systems?

Residual Neural Networks

\[ x(t + 1) = x(t) + F(x(t)) \]

Continuous-time Neural Networks

\[ \frac{dx}{dt} = f(x) \]
Neural Ordinary Differential Equations: A continuous-time NN

Neural Ordinary Differential Equations (Neural ODEs)

\[
\frac{dz}{dt} = f(z)
\]

Transformation of the hidden state

Neural ODEs model neural computation using a set of ordinary differential equations

\[
\frac{dz(t)}{dt} = NN(z, \theta, t)
\]

\(f\) is parameterized as a Neural Network!
Neural ODEs: Supervised Learning

**Forward phase**

\[
\frac{dz(t)}{dt} = f(z, \theta, t) \quad \text{ODE solver}
\]

\[z(0) = x\]

**Augmented variables**

\[z_{aug} = [z \quad \theta \quad t]^T\]

**Parameter gradients**

**Loss calculation**

\[
\begin{bmatrix}
\frac{d}{dt} \frac{\partial L}{\partial z} & \frac{d}{dt} \frac{\partial L}{\partial \theta} & \frac{d}{dt} \frac{\partial L}{\partial t}
\end{bmatrix}
= -\begin{bmatrix}
a_z \frac{\partial f}{\partial z} & a_z \frac{\partial f}{\partial \theta} & a_z \frac{\partial f}{\partial t}
\end{bmatrix}(t)
\]

\[a(t) = \frac{\partial L}{\partial z}\]

\[
\frac{da(t)}{dt} = a(t)^T
\]

**Backpropagation using Adjoint sensitivity method**

\[
\begin{bmatrix}
\frac{da_z}{dt} & \frac{da_\theta}{dt} & \frac{da_t}{dt}
\end{bmatrix} = -\begin{bmatrix}
a_z \frac{\partial f}{\partial z} & a_z \frac{\partial f}{\partial \theta} & a_z \frac{\partial f}{\partial t}
\end{bmatrix}(t)
\]

Compute gradients by solving the Initial Value Problem (IVP) starting from zero initial gradients:

\[
= [f(z, \theta, t) \quad 0 \quad 1]^T
\]

\[
\begin{bmatrix}
\frac{d}{dt} \frac{\partial L}{\partial z} & \frac{d}{dt} \frac{\partial L}{\partial \theta} & \frac{d}{dt} \frac{\partial L}{\partial t}
\end{bmatrix}
= -\begin{bmatrix}
a_z \frac{\partial f}{\partial z} & a_z \frac{\partial f}{\partial \theta} & a_z \frac{\partial f}{\partial t}
\end{bmatrix}(t)
\]


Neural CRNs

We have seen the operation of Neural ODEs! Can we reliably mimic their dynamics in chemistry? Is it possible to calculate precise gradients? What kind of functions can we able to represent? How difficult is the conversion?
Construction of Neural CRNs: Assumptions

1. Neural ODE of dimension 2
   \( z, a \in \mathbb{R}^2 \) and \( \theta \in \mathbb{R}^{2 \times 2} \)

2. \( f(z, \theta, t) = \theta z \)
   A linear neural network

3. Chemical species in dual-rail format
   \( x := [X_p] - [X_n] \)

4. Reactions written without a rate constant are unit rate constant.
Neural CRNs Construction: Type o ODE

\[
\frac{dx}{dt} = yz \quad x, y, z \in \mathbb{R}_{\geq 0}
\]

\[
Y + Z \rightarrow X + Y + Z
\]

Catalytic
Neural CRNs Construction: Type I ODE

\[ \frac{dx}{dt} = yz \quad x, y, z \in \mathbb{R} \]

\[ \frac{d(x_p - x_n)}{dt} = (y_p - y_n)(z_p - z_n) \quad x_p, x_n, y_p, y_n, z_p, z_n \in \mathbb{R}_{\geq 0} \]

- **Convert to "dual-rail" format**

- **Sequester**

- **Translate to CRNs**

\[ Y_p + Z_p \rightarrow X_p + Y_p + Z_p \]
\[ Y_n + Z_n \rightarrow X_p + Y_n + Z_n \]
\[ Y_p + Z_n \rightarrow X_n + Y_p + Z_n \]
\[ Y_n + Z_p \rightarrow X_n + Y_n + Z_p \]
Neural CRNs Construction: \textit{Type II ODE}

\[ \frac{dx}{dt} = -yz \quad x, y, z \in \mathbb{R} \]

\[ \frac{d(x_p - x_n)}{dt} = -(y_p - y_n)(z_p - z_n) \quad x_p, x_n, y_p, y_n, z_p, z_n \in \mathbb{R}_{\geq 0} \]

Convert to “\textit{dual-rail}” format

\[ \frac{dx_p}{dt} = y_p z_n + y_n z_p \quad \text{Type o ODE} \quad \frac{dx_n}{dt} = y_p z_p + y_n z_n \quad \text{Type o ODE} \]

Sequester

Translate to CRNs

\[ Y_p + Z_p \rightarrow X_n + Y_p + Z_p \]
\[ Y_n + Z_n \rightarrow X_n + Y_n + Z_n \]
\[ Y_p + Z_n \rightarrow X_p + Y_p + Z_n \]
\[ Y_n + Z_p \rightarrow X_p + Y_n + Z_p \]
CRNs for Addition, Subtraction & Multiplication

**Addition**

\[ y = a + b \]

- **crn_add**
  - \( A_p \rightarrow Y_p \)
  - \( B_p \rightarrow Y_p \)
  - \( A_n \rightarrow Y_n \)
  - \( B_n \rightarrow Y_n \)

**Subtraction**

\[ y = a - b \]

- **crn_subtract**
  - \( A_p \rightarrow Y_p \)
  - \( B_p \rightarrow Y_n \)
  - \( A_n \rightarrow Y_n \)
  - \( B_n \rightarrow Y_p \)

**Multiplication**

\[ y = a \cdot b \]

- **crn_mult**
  - \( A_p + B_p \rightarrow Y_p + A_p + B_p \)
  - \( A_n + B_n \rightarrow Y_p + A_n + B_n \)
  - \( A_p + B_n \rightarrow Y_n + A_p + B_n \)
  - \( A_n + B_p \rightarrow Y_n + A_n + B_p \)
  - \( Y_p \rightarrow 0 \)
  - \( Y_n \rightarrow 0 \)

\[ a, b, y \in \mathbb{R} \]
Assumption 2: $f(z, \theta, t) = \theta z$

A linear neural network

Forward Phase

Neural CRN forward

Output calculation

\[
\frac{dz}{dt} = f(z, \theta, t) = \theta z
\]

\[
\frac{dz_1}{dt} = \theta_{11} z_1 + \theta_{12} z_2
\]

Type I ODE

\[
\frac{dz_2}{dt} = \theta_{21} z_1 + \theta_{22} z_2
\]

Type I ODE

Feedforward CRN (partial)

\[
Z_{1p} + P_{11p} \xrightarrow{1.0} Z_{1p} + Z_{1p} + P_{11p}
\]

\[
Z_{1n} + P_{11n} \xrightarrow{1.0} Z_{1p} + Z_{1n} + P_{11n}
\]
Translating Neural ODEs to Neural CRNs

Forward Phase

Neural CRN forward

Output calculation

\[ \hat{y} = w_1 z_1 + w_2 z_2 \]

\[ w_1, z_1, w_2, z_2 \in \mathbb{R} \]

\[ \text{crn}_\text{add} \]

\[ \text{crn}_\text{mult} \quad \text{crn}_\text{mult} \]

Reminder: All values are calculated in the dual-rail format
Constructing Neural CRNs: Supervised Learning

\[ \frac{\partial L}{\partial w_i} = (\hat{y} - y) \frac{d\hat{y}}{dw_i} \]

\[ L = \frac{1}{2}(\hat{y} - y)^2 \]

\[ \hat{y} = w_1 z_1 + w_2 z_2 \]

- **Backprop**
- **Output layer gradients**
- **Hidden state backpropagation**
- **Adjoint backpropagation**
- **Calculating gradients**
- **Updating weights**
Constructing Neural CRNs

Recall

\[
\frac{dz}{dt} = -f(z, \theta, t) \quad \frac{dz(t)}{dt} = -f(z, \theta, t)
\]

\[
\begin{align*}
\frac{dz_1}{dt} &= -\theta_{11} z_1 - \theta_{12} z_2 \\
\frac{dz_2}{dt} &= -\theta_{21} z_1 - \theta_{22} z_2
\end{align*}
\]

Type II ODE  Type II ODE  Type II ODE  Type II ODE

\[
\begin{align*}
\frac{dz_1}{dt} &= -\theta_{11} z_1 \\
\frac{dz_2}{dt} &= -\theta_{21} z_1 - \theta_{22} z_2
\end{align*}
\]

\[
Z_{1n} + P_{11p} \xrightarrow{1.0} Z_{1p} + Z_{1n} + P_{11p} \\
Z_{1p} + P_{11n} \xrightarrow{1.0} Z_{1p} + Z_{1p} + P_{11n}
\]
Constructing Neural CRNs

Recall that in Neural ODEs

\[
\frac{da(t)}{dt} = a(t)^T \frac{\partial f(z, \theta, t)}{\partial z}
\]

\[
a(t) = [a_1 \quad a_2]^T \frac{\partial f(z, \theta, t)}{\partial z} = \theta
\]

\[
\frac{da_1}{dt} = \theta_{11} a_1 + \theta_{12} a_2
\]

\[
\frac{da_2}{dt} = \theta_{21} a_1 + \theta_{22} a_2
\]

\[
A_{1p} + P_{11p} \xrightarrow{1.0} A_{1p} + A_{1p} + P_{11p}
\]

\[
A_{1n} + P_{11n} \xrightarrow{1.0} A_{1p} + A_{1n} + P_{11n}
\]
Constructing Neural CRNs

Recall that in Neural ODEs

\[
\frac{d}{dt} \begin{bmatrix} g_{11} & g_{12} & g_{21} & g_{22} \end{bmatrix} = \begin{bmatrix} a_1 z_1 & a_1 z_2 & a_2 z_1 & a_2 z_2 \end{bmatrix}
\]

\[
\frac{d}{dt} \frac{\partial L}{\partial \theta} = a(t)^T \frac{\partial f}{\partial \theta}
\]

\[
\frac{\partial f}{\partial \theta} = \begin{bmatrix} z_1 & z_2 & 0 & 0 \\
0 & 0 & z_1 & z_2 \end{bmatrix}
\]

\[
A_{1p} + Z_{1p} \xrightarrow{1.0} G_{11p} + A_{1p} + Z_{1p}
\]

\[
A_{1n} + Z_{1n} \xrightarrow{1.0} G_{11p} + A_{1n} + Z_{1n}
\]
Constructing Neural CRNs

Gradient descent

\[ p^{new} = p^{old} - \eta g \]

Dual-rail format

\[ p_p^{new} - p_n^{new} = p_p^{old} - p_n^{old} - \eta g_p + \eta g_n \]
\[ p_p^{new} = p_p^{old} + \eta g_n \]
\[ p_n^{new} = p_n^{old} + \eta g_p \]

\[ G_n \xrightarrow{k_1} P_p \quad G_n \xrightarrow{k_2} \Phi \]
\[ G_p \xrightarrow{k_1} P_n \quad G_p \xrightarrow{k_2} \Phi \]

- \( p^{old} \): old param value
- \( p^{new} \): updated param value
- \( \eta \): learning rate
- \( g \): gradient value

\[ k_1 = \frac{\eta}{1 + \eta} \]
\[ k_2 = \frac{1}{1 + \eta} \]
Supervised Learning in Neural CRNs

Linearly separable binary classification dataset

#Training samples \((N) = 50\)

Predictions and Classification

Accuracy: 0.951920
Comparison of Neural CRNs with its Parent Neural ODE

Parameter value vs. Step

Gradient vs. Step
Extensions I: Nonlinear Separation Boundaries

Rings dataset, with a nonlinear separation boundary

#Samples ($N$) = 100

"Lifting" using the kernel trick

- The square ends are the starting points of the flow
- Notice that in 3D, the original points are linearly separable due to lifting.

Augmented Neural ODEs

Extension 2: Approximating a ReLU Network

IVP that approximates a `smoothed ReLU’

\[
\frac{dz_i}{dt} = h + (\theta_{i1} x_1 + \theta_{i2} x_2 + \beta_i) z_i - z_i^2
\]


**ReLU Network**

**Forward CRN**

\[
\begin{align*}
H & \rightarrow H + Z_i \\
P_i + X_1 + Z_i & \rightarrow 2Z_i \\
P_i + X_2 + Z_i & \rightarrow 2Z_i \\
B_i + Z_i & \rightarrow 2Z_i \\
2Z_i & \rightarrow \Phi
\end{align*}
\]

**Backpropagation CRN**

\[
\begin{align*}
\frac{da(t)}{dt} &= a(t)^T \frac{\partial f}{\partial z} = \begin{bmatrix} a_1 \theta_{11} x_1 + a_1 \theta_{12} x_2 - 2a_1 z_1 \\ a_2 \theta_{21} x_1 + a_2 \theta_{22} x_2 - 2a_2 z_2 \end{bmatrix} \\
\frac{d}{dt} \frac{\partial L}{\partial \theta} &= a(t)^T \frac{\partial f}{\partial \theta} = \begin{bmatrix} a_1 x_1 z_1 & a_1 x_2 z_1 & a_2 x_1 z_2 & a_2 x_2 z_2 \end{bmatrix} \\
\frac{d}{dt} \frac{\partial L}{\partial \beta} &= a(t)^T \frac{\partial f}{\partial \beta} = \begin{bmatrix} a_1 z_1 & a_2 z_2 \end{bmatrix}
\end{align*}
\]

**smoothed ReLU**

\[
\frac{1}{2} (y + \sqrt{y^2 + 4h})
\]

**Tri-molecular reactions**
Extension 3: Approximating Multi-layer Networks

- The **adjoint, hidden state** at the end of backpropagation from $2T \rightarrow T$ are used as initial values for backpropagation from $T \rightarrow 0$

- The backpropagation in Neural CRN1 and Neural CRN2 are independent of each other

- Therefore, backpropagation scales **linearly**!
Neural CRNs: Natural Implementation of Learning in Chemical Reaction Networks

Rajiv Teja Nagipogu
Department of Computer Science, Duke University, Durham, North Carolina, USA

John H. Reif
Department of Computer Science, Duke University, Durham, North Carolina, USA

Abstract

We present Neural CRNs: a set of deterministic chemical reaction networks (CRNs) that work in tandem to simulate autonomous learning in the chemical medium. Molecular circuits capable of learning and adaptation in uncertain chemical environments could engender novel applications in fields such as synthetic biology, biotechnology, and medicine. They could be employed for tasks that require sensing and actuation at the microscopic scale (e.g., targeted drug delivery). In this regard, existing implementations of molecular-scale learning have mostly focused on instantiating chemical equivalents of standard neural networks such as the perceptron and, more recently, multi-layer neural networks. However, such implementations scale poorly due to the inherent incompatibility between digital and chemical computing paradigms—digital computing is discrete, whereas chemical computation is continuous—and could lead to large and cumbersome reaction network implementations. For example, most previous works settle with computing first-order approximations of gradients in their chemical implementation of the backpropagation algorithm. In pursuit of chemically compatible learning implementations, we look towards dynamical system-based approaches for neural learning. In particular, we examine a specific example, Neural Ordinary Differential Equations (Neural ODEs), which models neural computations with a set of ordinary differential equations (ODEs). We consider a simple version of the Neural ODE framework by instantiating its dynamics function with a linear neural network and systematically translating the resultant ODE system into a set of CRNs, resulting in our Neural CRN framework. We then demonstrate several examples of training and classification by Neural CRNs. We also show that the operation of Neural CRNs matches closely with its parent Neural ODE by comparing the trajectories of their parameters and gradients while training on a binary classification task. Finally, we discuss several extensions to Neural CRNs, including approximating nonlinear separation boundaries, incorporating nonlinear activation functions (ReLU), and assembling multi-layer networks. The code to reproduce our simulations can be found here.
Summary

- DNA Computing & Molecular Computing
- Learning and Advantages of Molecular scale learning
- Literature Review
- ODEs of Neural ODEs
- Constructing Neural CRNs
  - Preliminaries
  - Forward phase CRNs
  - Backpropagation phase CRNs
  - Weight update CRNs
- Training and Classification by Neural CRNs
- Comparison of Neural CRNs w/ parent Neural ODE
- Extensions
  1. Nonlinear separation boundary
  2. ReLU activation networks
  3. Multi-layered networks

Thanks!

Questions?
Input $\rightarrow$ Feedforward Neural CRN ($Z$) $\rightarrow$ Feedforward Output layer ($\hat{Y}$) $\rightarrow$ Error calculation ($\hat{Y} - Y$)

Update Weights

- Backprop Hidden state ($Z$)
- Backprop Adjoint ($A$)
- Backprop Calc. Gradients ($G$)

Gradients of output layer ($WG$)

Initial Adjoint $A(t_f)$