

ディジタルリングリザバーと そのオンライン学習器の 統合アーキテクチャに関する研究

A study on hardware integration of
a digital ring reservoir and its online learning system

2023年4月

(暫定版)

北海道大学
大学院情報科学院
情報科学専攻 情報エレクトロニクスコース
集積システム講座 集積ナノシステム研究室

ディジタルリングリザバーとそのオンライン学習器の 統合アーキテクチャに関する研究

要旨

本研究は、時系列データを扱う機械学習モデルとして近年注目されている Reservoir Computing のハードウェアアーキテクチャに関するものである。Reservoir Computing は、再帰的な結合を持つニューラルネットワークの一種であり、学習手法により軽量な学習を可能にしたモデルである。リソースが限定期的なハードウェアにおいて推論と学習を行う時系列 AI を実装するためには軽量な学習手法を持つモデルが不可欠であり、Reservoir Computing はこの用途に最適な機械学習モデルである。

本研究の目的はReservoir Computingをより実応用に近づけるという点にある。学習と推論がエッジで完結する Reservoir Computing デバイスは未だほとんど存在していない。その障壁となっているのは、リザバー部と学習部（Reservoir Computing を構成する 2つの要素）を 1 つのデバイスに実装する際のリソース的制約である。本研究では、従来の一斉更新型のリザバーネットワーク (Echo State Network) に時分割多重方式を取り入れ、ネットワークを小型化したモデルを考案し FPGA アーキテクチャを構築した。その結果、従来の数百分の 1 規模に回路を縮小することができ、それまで困難であった安価な FPGA へ実用的なリザバーを実装することが可能となった。これにより、先行研究により実装されていた FPGA FORCE 学習器と 1 つの FPGA に実装することが可能となり、1 つのデバイスで完結する FPGA Reservoir Computer が実現した。

目 次

第 1 章 Reservoir Computing	9
1.1	3
1.1 原理	9
1.1.1 Echo State Network	9
1.1.2 Echo State Property	11
1.2 学習手法	12
1.2.1 線形回帰	12
1.2.2 FORCE Learning	14
1.3 評価指標	15
1.3.1 NARMA10	15
1.3.2 Memory Capacity	16
1.3.3 Informational Processing Capacity	17
参考文献	18
第 2 章 Ring Reservoir アーキテクチャの考察	19
2.1 Ring Reservoir	19
2.2 FPGA アーキテクチャ	23
2.2.1 シフトレジスタ型	23
2.2.2 RAM 型	24
2.2.3 Piecewise Linear Function による省メモリ活性化関数	27
2.3 学習機との統合	28

2.3.1	シングルコア FORCE Learning Accelerator について	28
2.3.2	統合	30
2.4	評価と考察	32
2.4.1	Ring Reservoir アーキテクチャの評価	32
2.4.2	Piecewise Linear Function による性能変化	35
2.4.3	学習機との統合評価	39

第1章 Reservoir Computing

1.1 原理

1.1.1 Echo State Network

Reservoir Computing は、2001 年に Wolfgang Maass 氏が提案した Liquid State Machines(LSM) [1] と Herbert Jaeger 氏が提案した Echo State Network(ESN) [2] が基になっている [3]。

本研究は、後者の ESN を土台としており、本章では、その原理や学習手法、評価指標について説明する。

ESN は一般に図 1.1 のような図で表現され、中間層の結合重みを固定した Recurrent Neural Network(RNN) を用いて、時系列入力の過去の情報が反響して残る状態 (Echo State) を作り出し、そこから入力の特徴の読み出し (Readout) を行う [4]。また、Readout の結合重みの学習手法については、次項で詳細を説明する。

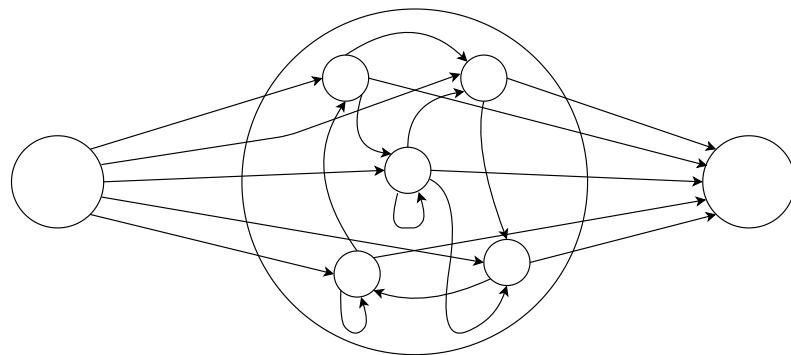


図 1.1 Echo State Network

ESN における時刻 $t+1$ でのノード x_i の更新式は以下の式で表される。なお、 w_{in} はノード i に対する入力結合重み、 $u(t)$ は時刻 t におけるリザバーへの入力値、 N はノード数、 w_{ij} はノード i に対するノード j の結合重み、 b はバイアス項、 f は活性化関数である。一般に、活性化関数には \tanh が用いられることが多い。

$$x_i(t+1) = f \left\{ w_{in} u(t+1) + \sum_{j=0}^{N-1} w_{ij} x_j(t) + b \right\} \quad (1.1)$$

Reservoir Computing および ESN は、RNN(図 1.2) の一種と分類されるが、通常の RNN との違いは学習の対象範囲と学習手法である。RNN は、時間方向に展開した中間層同士を結合する重みも含めてネットワーク内の全ての重みを学習対象とするため、誤差を出力層から伝搬する BPTT(Back Propagation Through Time) で学習を行うのが一般的である。一方で ESN では、中間層と出力層を結合する重みのみを学習対象とし、入力層と中間層、中間層同士を結合する重みは乱数などで学習前に決定し更新せずに固定の重みとする。それにより、ESN では RNN の BPTT に比べ非常に軽量な学習手法である線形回帰で学習が可能である [5]。

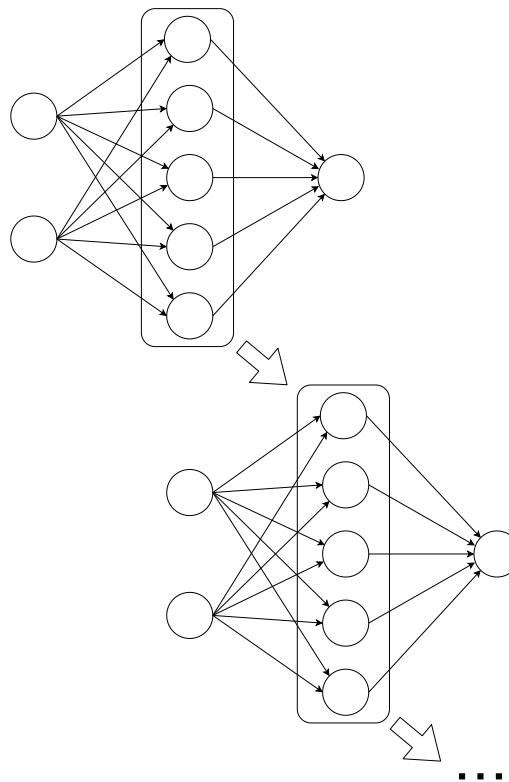


図 1.2 Recurrent Neural Network

RNN に対する ESN の他の利点も含めると、それぞれの特徴は下の表のようにまとめることができる。また、Reservoir Computing では、入力層からリザバー部までの構造は学習前後で不变であるため、作成した構造から変更を加えないという点でハードウェアに向いていると言うことも出来る。

表 1.1 RNN に対する ESN の利点

RNN	ESN
複雑で重負荷な学習手法	シンプルで軽量な学習手法
最適な重みを見つけるのが難しい	最適な重みが一意に決定
追加学習によって干渉	追加学習が容易

1.1.2 Echo State Property

Reservoir Computing は、その構造と学習方法を用いれば必ず期待される動作をするわけではなく、学習可能にさせるための条件が存在する。その条件とは、制御することなく再現性を持つということである。この特性は Echo State Property(ESP) と呼ばれ、Reservoir Computing を構成する上で最も重要な要素のひとつである。

具体的には、構造が全く同じで初期値が異なる Reservoir Computing を 2 つ用意し、それらに同じ入力を与え続けたときに最終的に初期値の影響をなくすことができ、出力値が全く同じ値に収束できるかどうかということである。ここで、力学系における軌道から離れていく度合いを表すリヤプノフ指数 (Lyapunov exponent) を用いれば、この条件は値が収束していくことがあるので $\lambda < 0$ を満たすということと同値であることがわかる。なお、 $\lambda > 0$ では、再現性を持たずに予測できない値の変化を起こすカオス (Chaos) の状態に転移する。カオス下では初期値鋭敏性 (バタフライ効果) が現れ、同じ系であっても初期値が少し異なるだけで時間経過後それが大きな差になる。これが故、基本的にカオス下では ESP を持たないが、カオスは多様性に富んだダイナミクスを生むとされているため、カオスを抑制し Reservoir Computing に応用する研究も行われている。

$\lambda > 0$ が望まれる一方で、 $\lambda \ll 0$ であると、収束するのが早く、過去の入力情報の影響をほとんど受けなくなってしまう。よって、過去の情報を忘れにくく、かつ再現性を得るためににはリヤプノフ指数 λ を 0 近い負の値にする必要がある。つまり、カオスに近いがそうではない値、Edge of Chaos に調整する必要がある (図 1.3)。

ESNにおいて、この Edge of Chaos に近づけるためには、リザバー内部の重み行列のスペクトル半径を 1 に満たない 1 に近い値に事前に調整することで可能となる。学習によってリザバー内部の重みは更新しないが、このように事前に重み行列を調節することは事前トレーニング (Pretraining) と呼ばれる。

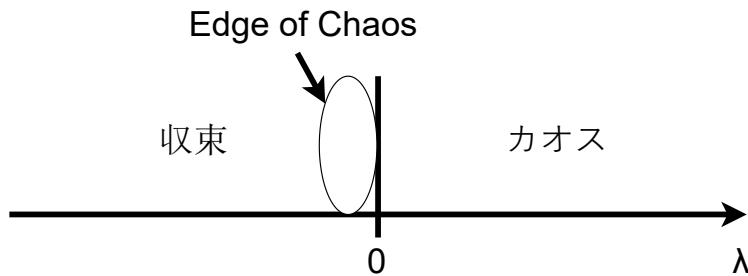


図 1.3 Edge of Chaos

1.2 学習手法

1.2.1 線形回帰

本項では、ESN での線形回帰による学習手法を説明する。

図1.4 のように、それぞれのニューロン(ノード)の状態を x_0, x_1, \dots, x_{N-1} と置き、それぞれのノードの状態と出力層を結合する重みを w_0, w_1, \dots, w_N と置く。ESN では、出力層の活性化関数は非線形関数ではなく線形和を取るのが一般的であり、バイアス項を b とすると ESN の出力 y は次式になる。

$$y = w_0x_0 + w_1x_1 + \dots + w_{N-1}x_{N-1} + b \quad (1.2)$$

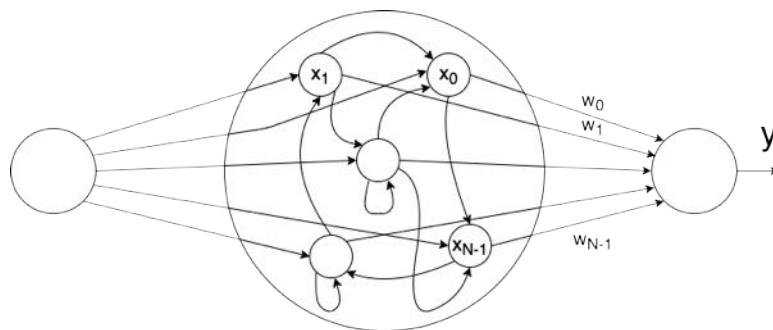


図 1.4 ノードの更新および学習の計算

さらに、バイアス項 b を常に 1 を出すような定常ノード x_N の出力結合重みと見立てると以下のようになる。

$$y = w_0x_0 + w_1x_1 + \dots + w_{N-1}x_{N-1} + w_Nx_N \quad (1.3)$$

$$= w_0x_0 + w_1x_1 + \dots + w_Nx_N \quad (1.4)$$

ここで、 t 秒後の出力 y を y_t として、 T 秒後までの出力をベクトルにまとめると以下のように表現できる。

$$Y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_T \end{pmatrix} \quad (1.5)$$

また、リザバー(中間層)のそれぞれのノードの状態 x_0, x_1, \dots, x_N を時間方向に展開し、 t 秒後のノードの状態を $x_{0,t}, x_{1,t}, \dots, x_{N,t}$ として、 T 秒後までのそれらを行列 X にまとめると次のようになる。

$$X = \begin{pmatrix} x_{0,0} & x_{1,0} & \dots & x_{N,0} \\ x_{0,1} & x_{1,1} & \dots & x_{N,1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{0,T} & x_{1,T} & \dots & x_{N,T} \end{pmatrix} \quad (1.6)$$

また、重みは時間で不变なので次のようなベクトルにまとめられる。

$$w = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{pmatrix} \quad (1.7)$$

上記2式より、出力ベクトル Y は次式のように表すことができる。

$$Y = Xw \quad (1.8)$$

学習計算とは教師データに対して最適な重みを求めることがあるので、上式から最小二乗法による線型回帰で w を求めることを考える。

Y を教師データに置きかえると、誤差 J は

$$J = \|wX - Y\| \quad (1.9)$$

となり、この誤差 J が最小となる条件は

$$\frac{\partial J}{\partial w} = 0 \quad (1.10)$$

である。上の 2 式より w の学習は次の行列変換で表現できる。

$$w = ({^t}XX)^{-1}{^t}XY \quad (1.11)$$

X は正則ではないため逆行列は存在しないが、存在するとして計算すると、

$$w = X^{-1}Y \quad (1.12)$$

となり、式 (1.7) の左側から X^{-1} を掛け合わせた形になる。また、これは Y に一般化逆行列(擬似逆行列)を左からかけることで w が求まることも示している。

このように、ESN では学習を行列演算のみで行うことができ、リザバー内部の状態を保存しておくことで、その行列と教師信号のベクトルのみで一意に重みを決定することができる。この学習手法は、複雑で計算コストが大きい BPTT より非常にシンプルで軽量な計算が可能である。

1.2.2 FORCE Learning

FORCE(First Order Reduced and Controlled Error)Learning [6] は、ESN を拡張したアルゴリズムである。一般的な ESN にフィードバック結合が追加されたネットワークにオンライン学習を適応させたアルゴリズムであり、線形回帰と比較して様々な利点を持つ。本来は ESN の特殊系であるネットワークとオンライン学習を包括して FORCE 学習と呼ばれるが、このオンライン学習は様々なネットワークに適応することが可能であるため、ここでは任意のネットワークにオンライン学習を適応することを FORCE 学習と定義する。

FORCE 学習とは、逐次最小二乗法に基づくオンライン学習手法である。線形回帰での学習には逆行列演算が必要であるが、逐次最小二乗法を用いることで逆行列演算に相当する以下の更新式を計算することで毎学習ステップの結合重みを回帰的に計算することができる。時間 t における自己相関行列を $N \times N$ 次元の行列 $\hat{P}(t)$ 、リザーバの出力を N 次元のベクトル $\vec{R}(t)$ 、重みを N 次元のベクトル $\vec{W}(t)$ 、出力誤差を $e(t)$ とすると、更新式は次式で表される。

$$Z(t) = \vec{W}^T(t-1)\vec{R}(t) \quad (1.13)$$

$$e(t) = Z(t) - S(t) \quad (1.14)$$

$$\hat{P}(t) = \hat{P}(t-1) - \frac{\hat{P}(t-1)\vec{R}(t)\vec{R}^T(t)\hat{P}(t-1)}{1 + \vec{R}^T(t)\hat{P}(t-1)\vec{R}(t)} \quad (1.15)$$

$$\vec{W}(t) = \vec{W}(t-1) - e(t)\hat{P}(t)\vec{R}(t) \quad (1.16)$$

線形回帰による学習と FORCE 学習の特徴をまとめたのが以下の表である。

表 1.2 線形回帰と FORCE 学習の比較

線形回帰	FORCE 学習
バッチ学習	オンライン学習
(学習時間) × (ノード数) のメモリが必要	(ノード数) × (ノード数) のメモリが必要
逆行列演算が必要	単純な行列演算で計算可能

1.3 評価指標

1.3.1 NARMA10

まず、Reservoir Computing の代表的な評価指標である NARMA10(10th-order Non-linear Auto-Regressive Moving Average) タスク [7] の NRMSE(Normalized Root Mean Square Error, 正規化平方平均二乗誤差) を説明する。

NARMA10 は、10 ステップ前までの値を参照して現在の値を更新する関数であり、Reservoir Computing によるその関数の再現度を教師データと推論値の誤差 (NRMSE) を算出することで測る。

以下に NARMA10 の式を示す。

$$y(t+1) = 0.3y(t) + 0.05y(t) \left\{ \sum_{k=0}^9 y(t-k) \right\} + 1.5u(t)u(t-9) + 0.1 \quad (1.17)$$

なお、Reservoir Computing の評価時は $y(t+1)$ が教師信号にあたり、 $u(t)$ は入力信号にあたり、入力は $0 \leq u \leq 0.5$ の一様乱数である。

また、NRMSE は以下の式の通りである。この式の $y_{1,t}$ と $y_{2,t}$ が Reservoir Computing の出力値と教師信号に、 T が学習ステップ数にあたる。

$$RMSE = \sqrt{\frac{\sum_{t=1}^T (y_{1,t} - y_{2,t})^2}{T}} \quad (1.18)$$

$$NRMSE = \frac{RMSE}{\bar{y}} \quad (1.19)$$

1.3.2 Memory Capacity

MC(Memory Capacity) は、リザバーがどれだけの時間、入力の影響を保持できるのかという指標である。図 1.5 のように、 $u(n)$ の入力に対して z の出力が出る Reservoir Computing があるとする。このとき τ 秒前の入力を教師信号として学習して、推論したときに推論値がどれだけ教師信号と相関関係があるかを入力に対してそれぞれ求めていき、その値を合算する。 τ 個前の入力に対する相関関係 MC_τ は以下の式で求められる [8]。

$$MC_\tau = \frac{cov^2(u(t-\tau), z)}{\sigma^2(u(t-\tau))\sigma^2(z)} \quad (1.20)$$

さらに MC は次式で計算される。

$$MC = \sum_{\tau=0}^{\infty} MC_\tau \quad (1.21)$$

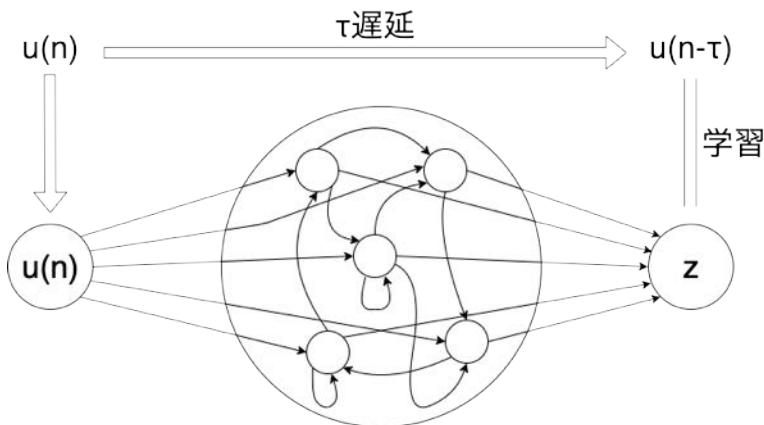


図 1.5 Memory Capacity

1.3.3 Informational Processing Capacity

Informational Processing Capacity(IPC)は、文献 [9] で提案されている、ルジャンドル多項式(式 (1.22))を基にした式 (1.23)の教師信号でリザバーの非線形容量を測定する評価方法である。なお、式 (1.23) 中の $\{d_i\}$ は次数の集合であり、 d_0, d_1, d_2, \dots を表す。

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n, n = 0, 1, 2, \dots \quad (1.22)$$

$$y_{\{d_i\}}(t) = \prod_i P_{d_i}(u(t - i)) \quad (1.23)$$

上記の教師信号を基に学習を行い、Reservoir Computing の出力値と教師信号から以下の式で各教師信号に対する Capacity を求める。T は評価時間を、第2項は NMSE(Normalized Mean Square Error, 正規化平均二乗誤差) を表している。

$$C_T = 1 - \frac{\min_w \text{MSE}[\hat{y}]}{\langle y^2 \rangle_T} \quad (1.24)$$

上記の式で求めた Capacity を以下のように変換する。なお、 $\theta(\cdot)$ はヘヴィサイド階段関数であり、 $C_T(\{d_i\}) \geq \epsilon$ の時に $C_T^\epsilon(\{d_i\}) = C_T(\{d_i\})$ 、それ以外で $C_T^\epsilon(\{d_i\}) = 0$ となる。これは、 ϵ で下限を設けることで、非常に小さい値の Capacity を無視し、計算が膨大になるのを避けるためである。

$$C_T^\epsilon(\{d_i\}) = \theta(C_T(\{d_i\}) - \epsilon) C_T(\{d_i\}) \quad (1.25)$$

上記の式の $\{d_i\}$ の総和を評価次数として、計算した $C_T^\epsilon(\{d_i\})$ をそれぞれの評価次数ごとに加算したのが、IPC における次数毎の Capacity となる。

参考文献

- [1] Maass Wolfgang, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 2002, 14.11: 2531-2560.
- [2] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 2001, 148.34: 13.
- [3] Lukoševičius Mantas, and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 2009, 3.3: 127-149.
- [4] 田中剛平, 中根了昌, 廣瀬明. リザバーコンピューティング. 2021.
- [5] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, Vol. 126, pp. 191-217, 2022.
- [6] D. Sussillo, and L. F. Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, Vol. 63, No. 4, pp. 544-557, 2009.
- [7] Herbert Jaeger. Adaptive nonlinear system identification with Echo state networks. *Proceedings of the 15th International Conference on Neural Information Processing Systems*, 2002.
- [8] Jaeger, Herbert. Short term memory in echo state networks. 2001.
- [9] Appeltant, Lennert, et al. Information processing using a single dynamical node as complex system. *Nature communications*, 2011, 2.1: 1-6.

第2章 Ring Reservoir アーキテクチャ の考案

2.1 Ring Reservoir

Reservoir Computing の代表なモデルである ESN は、リザバー内で Echo 状に入力情報を蓄積・変換することが可能である一方、ノード数分の活性化関数を必要とする。活性化関数には一般的に非線形関数を用いるため、ハードウェア実装を考えた際、活性化関数の数が多くなるほどハードウェアの限られたリソースを圧迫するという課題がある。そこで、活性化関数を 1 つだけ用意し、そこに時分割多重方式により入力信号を伝送することにより、ESN に近いリザバーを実現することが可能である。本論文では、この方式で構成されたリザバーを Ring Reservoir と呼ぶ。Ring Reservoir では、活性化関数を 1 つにすることができる他、それに伴いリザバー層の入力線・出力線を 1 つずつにすることが可能である。一方で、1 つの入力信号をノード数分に時分割して処理を行うため、一度に処理を行う ESN より計算速度は劣る。また、ネットワークの形が制限されてしまうというデメリットは生じる。

まず、Ring Reservoir の入力方式について説明する。

例として、ノード数 $N = 10$ を考える。入力結合重みは 10 個必要になるが、Ring Reservoir は時分割多重方式であるため、1 ステップ毎に 1 個の入力結合重みを参照する。したがって、1 サイクルの入力信号に対してノード数 N 時間分の 10 ステップが必要である。図 2.1 の 1 段目と 2 段目のような入力結合重み w_{in} と入力信号 in があるとすると、それらを掛け合わせた値は同図の 3 段目 (j) のようになる。このように、入力信号に対して w_{in} をマスクとして掛け合わせた j が活性化関数へ伝送される。

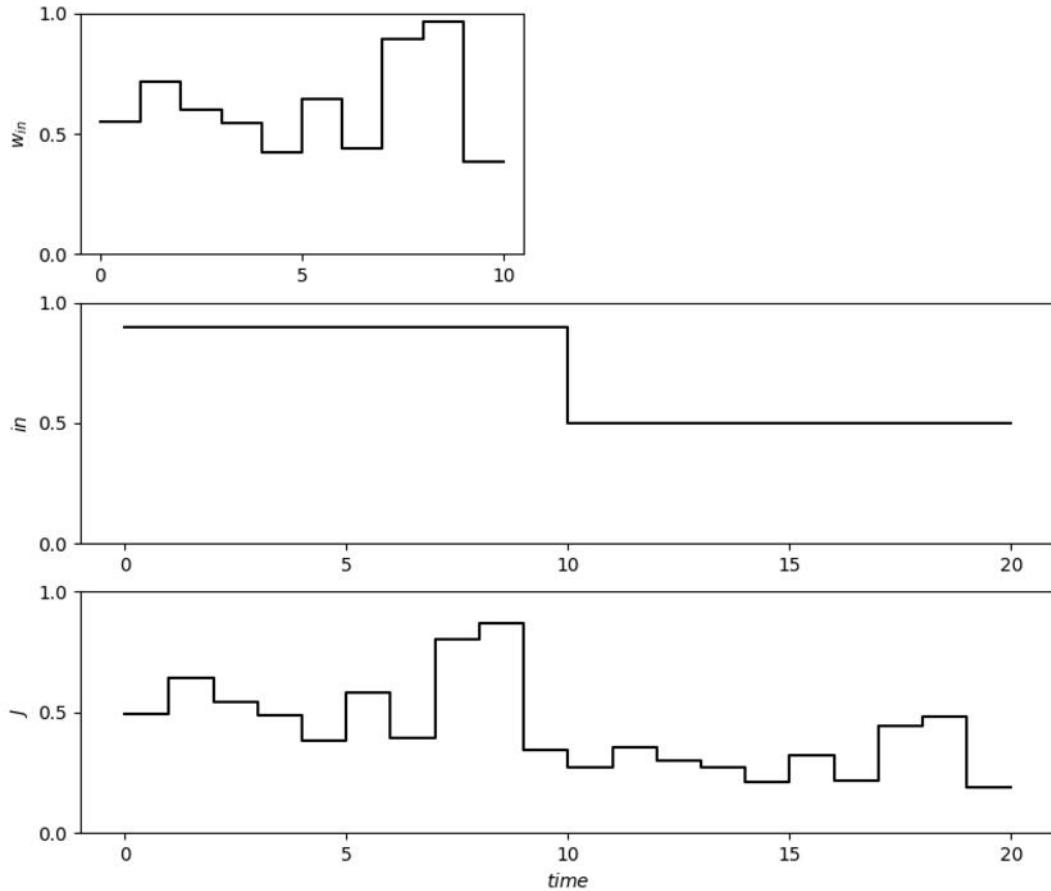


図 2.1 入力信号と結合重みの重ね合わせ

次に、1ノード分の更新について説明する。

Ring Reservoir は図 2.2 のような図で表される。リング状になっているノードの集合は仮想ノードと呼ばれ、単位ステップ毎にノードの値が矢印方向にシフトしていく、図の x_{vir0} に格納されている値は次のステップに内部結合重み w と掛け合わされて、前述の重み付された入力信号と同様に活性化関数の入力になる。活性化関数の出力はそれをノード値のひとつとして外部に出力するものに加え、仮想ノードの末端ノードに同じ値をコピーし格納する。

活性化関数を f とすると、図 2.2 における $y_i(t+1)$ は次式で表すことができる。なお、図 2.2 の外部ではこの $y_i(t+1)$ を更新されたひとつのノード値として扱うので、リザバー全体として見たときのノード i 版目の状態を x_i とすると $y_i(t+1) = x_i(t+1)$ である。

$$y_i(t+1) = f\{w_{in}u(t+1) + wx_{vir0}(t)\} = x_i(t+1) \quad (2.1)$$

これで、1ノード分の計算が完了する。全ノードの更新にはこれをノード数 N 回繰り返すことになり、それが終わると入力信号 1つに対するサイクルが完了したことになる。

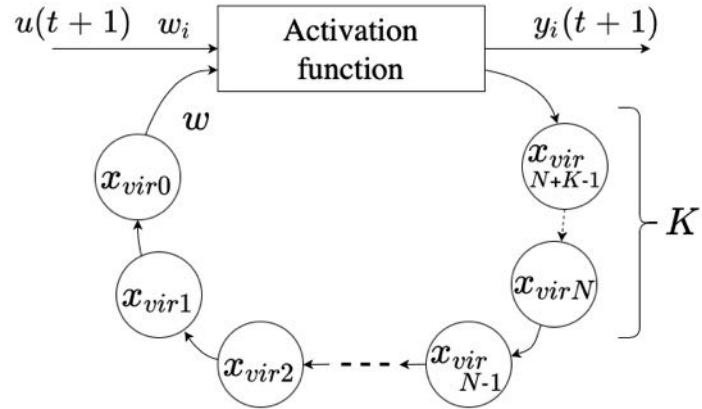


図 2.2 Ring Reservoir

次に、図中の仮想ノード数 $N + K$ 個の K について言及する。

式(2.1)より、リザバー全体として見たときのノード i の時刻 $t + 2$ のおける更新式は以下のようになる。

$$x_i(t+2) = f \{ w_{in}u(t+2) + wx_{vir0}(t+1) \} \quad (2.2)$$

$K = 0$ の場合、仮想ノード数が更新されるノード数(=リザバー全体として見たときのノード数) N と等しくなり、式(2.2)中の $x_{vir0}(t+1)$ はちょうど N ステップ前に更新され仮想ノードの最後尾にコピーされた値であるため、以下のようなになる。ここで、 x の添字が式(2.2)と同一であることがわかる。

$$x_{vir0}(t+1) = x_i(t+1) \quad (2.3)$$

式(2.3)を式(2.2)に代入すると、以下のようにになり、結合が自己回帰のみになっていることがわかる。

$$x_i(t+2) = f \{ w_{in}u(t+2) + wx_i(t+1) \} \quad (2.4)$$

このネットワークをリザバー全体として考えてみると、実質的には図(2.3)のようなネットワークになる。

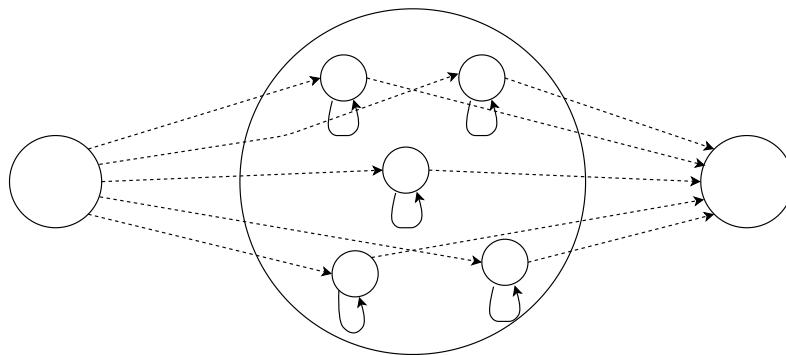


図 2.3 $K = 0$ の場合の実質的なネットワーク

このネットワークでは、Reservoir Computing 特有の複雑なダイナミクスを活用することができない。そこで、仮想ノードに K 個の余剰ノードを加えると、上記の式(2.3)、式(2.4)は次のように変わる。

$$x_{vir0}(t+1) = x_{i-K}(t+1) \quad (2.5)$$

$$x_i(t+2) = f \{ w_{in}u(t+2) + w x_{i-K}(t+1) \} \quad (2.6)$$

これにより、実質的なネットワークは図(2.4)のように変化し、自身ではない別なノードの情報を参照することができ、ESN に近い複雑なダイナミクスを生むリザバーとなる。

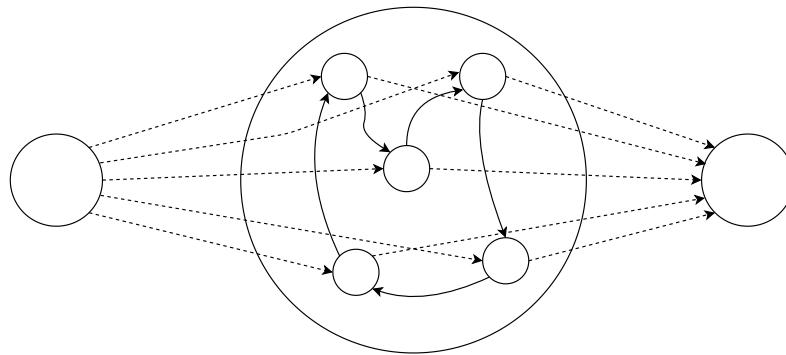


図 2.4 $K \neq 0$ の場合の実質的なネットワーク

2.2 FPGA アーキテクチャ

2.2.1 シフトレジスタ型

本研究では、ESN よりハードウェアに適した Ring Reservoir をさらに低リソースなハードウェアとして実装するために、効率的なアーキテクチャ検討し FPGA 実装とその評価を行った。

Ring Reservoir の FPGA アーキテクチャとして、本研究ではシフトレジスタ型と RAM 型の 2 つを構築した。本項では、はじめにシフトレジスタ型について説明する。

シフトレジスタ型の Ring Reservoir アーキテクチャとは、仮想ノードの部分をシフトレジスタを用いて構成したものであり、Ring Reservoir の概念図から直感的に動作が理解可能である。本研究で考案したシフトレジスタ型アーキテクチャを図 2.5 に示す。なお、この図では簡易的にノード数を $N = 3$ 、仮想ノード数を $N_{vir} = 4(K = 1)$ にしている。

以下、動作フローを説明する。

まず、ROM にはノード数分の入力結合重み W_{IN} がアドレスごとに格納されており、入力結合重み用アドレスカウンタからアドレスを受け取り、そのアドレスに応じた入力結合重みを出力する。入力 $I(t)$ はその入力結合重みと掛け合わされ、内部結合重み W と掛け合わされた仮想ノードの先頭 WX と加算される。その後、活性化関数(本研究では \tanh を用いる)で変換され出力される。その出力値は仮想ノードの最後尾 VIR_X0 にコピーされ、仮想ノードとして見立てたシフトレジスタを 1 クロックごとにシフトしていく。上記を 1 ノード分の更新として、それを入力 $I(t)$ を保ったまま NCLK 繰り返すことで、入力 1 サイクルに対する全てのノードの更新が完了する。なお、Readout として用いるノード値は仮想ノードを用いるのではなく、1CLK ごとに出力される本アーキテクチャの出力値をストリームで用いるか別途外部の RAM 等に保存して用いることを想定している。なお、パスが長い箇所(クリティカルパス)へはパイプラインレジスタを挟み、高周波動作時の不具合を防止した。

なお、活性化関数である \tanh の実装については後述する。

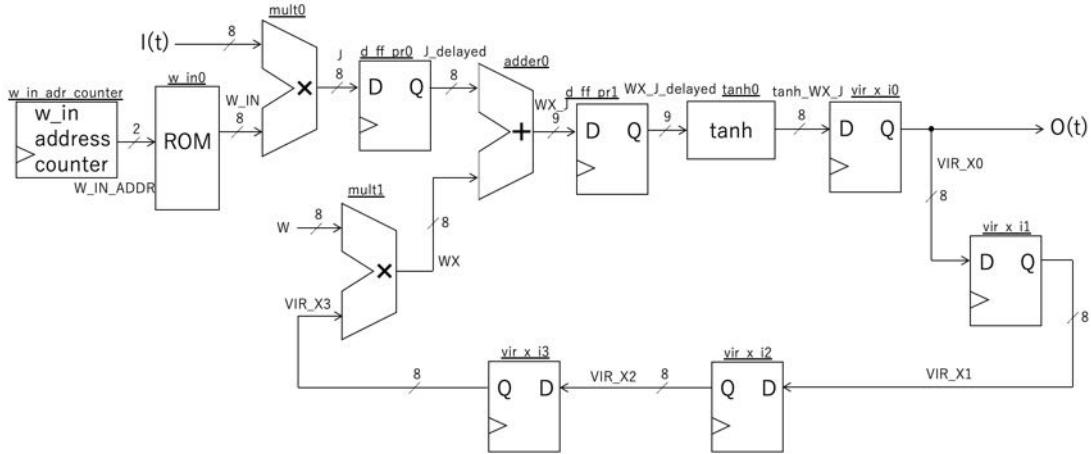


図 2.5 シフトレジスタ型 Ring Reservoir のアーキテクチャ

2.2.2 RAM 型

次に、RAM 型アーキテクチャを説明する。先述のシフトレジスタ型アーキテクチャの場合、仮想ノード数が大きくなるほど必要なレジスタ数が多くなる。実際に複雑なタスクを行うためには数百ほどのノード数を必要とするため、レジスタの数が非常に多くなってしまいアーキテクチャ的に効率的ではない。そのため、仮想ノードにシフトレジスタではなく RAM を用いることでその問題を解決した。シフトレジスタ型における仮想ノードの最後尾へのコピーの動作を、本アーキテクチャではカウントアップするアドレスごとに RAM へ書き込んでいき、RAM 内の一番古い情報を読み出していくことで、情報がリング状に転移していく仮想ノードを模倣した。つまり、RAM を FIFO(First In, First Out)で用いることで仮想ノードの動作を再現した。

本研究で考案した RAM 型 Ring Reservoir のアーキテクチャを図 2.6 に示す。この図もシフトレジスタ型と同様にノード数 $N = 3$ 、仮想ノード数 $N_{vir} = 4$ である。全体的なアーキテクチャの構造は、先述のシフトレジスタ型と変わりはないが、仮想ノードの部分をシフトレジスタから、RAM とアドレスカウンタに置き換えた。

RAM の動作としては単純で、アドレスカウンタから送られてきたアドレスに値を書き込むの同時に、そのアドレスに格納されていた古い情報を書き込まれる前に出力するというものである。こうすることで、RAM 内の一番古い情報を常に出力にすることができる、FIFO を実現することができる。

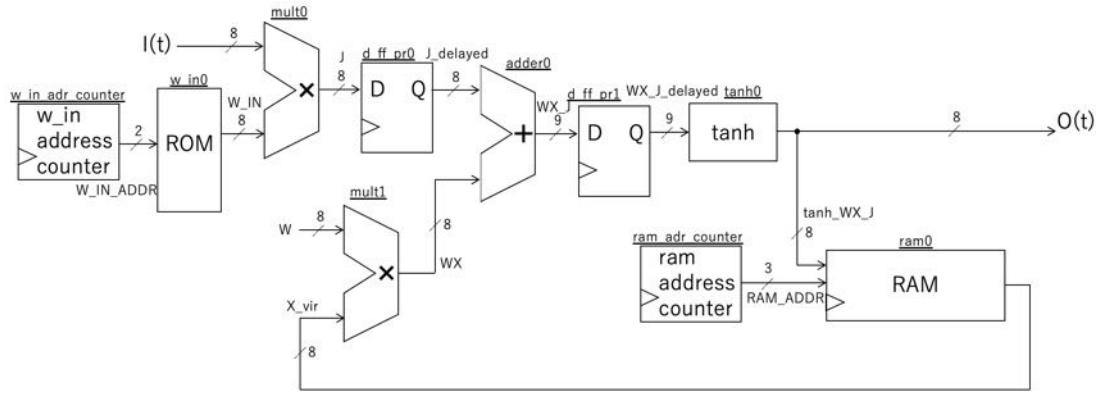


図 2.6 RAM 型 Ring Reservoir のアーキテクチャ

では、RAM 内部の値の変化と入力値に対してどの仮想ノードの値が使用されるか説明する。

図 2.7 における表は RAM 内部のアドレスとそれに対応したデータを表しており、1 入力後というのは 1 つの入力サイクルを意味しており、この図の場合 $N = 3$ であるので 3 クロック後を意味する。また、点線部は次の入力(3 クロック分)に使用される仮想ノードの値を表している。

図の左上は RAM の初期状態である。 $N_{vir} = 4$ であるので、アドレスは 0 から 3 まであり、それぞれに初期状態 A, B, C, D が格納されているとする。この状態では、次の入力サイクルに対応する値は A, B, C である。次に右上の 1 入力サイクルを終えた後の RAM の状態を見ると、このアーキテクチャでは RAM から出力した情報が処理され、1 周回って入力されるまでの遅延が 2 クロックあるため、前の入力での最初の値 A が変換された値 A' が RAM のアドレス 2 に書き込まれている。そして、さらに次の入力サイクル中にアドレス 3 と 0 と 1 が順に更新され、次の入力サイクルに対応したアドレスはアドレス 2,3,0 である。ここで、初期状態から入力サイクルで使用されたデータを見てみると、(A, B, C)、(D, A, B)、(A', B', C') と、元となるデータが遷移しており、他のノードと作用するダイナミクスを生むことがわかる。このようにして、RAM とアドレスカウンタを用いることでリング型の仮想ノードを再現することが可能である。

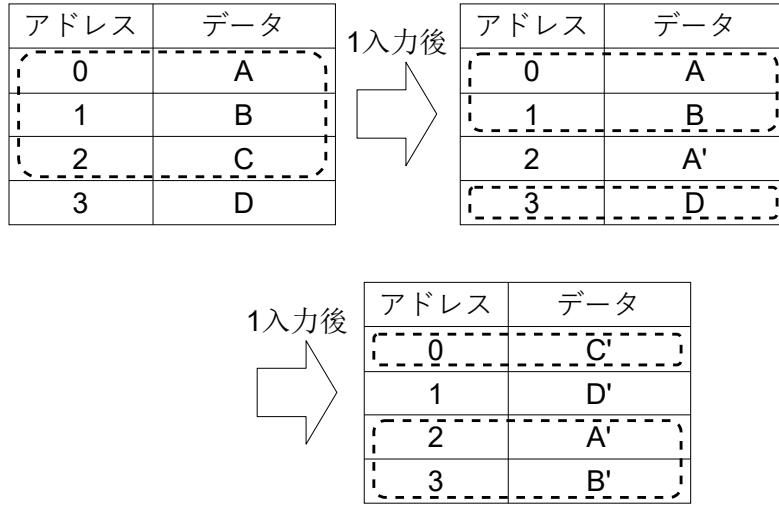
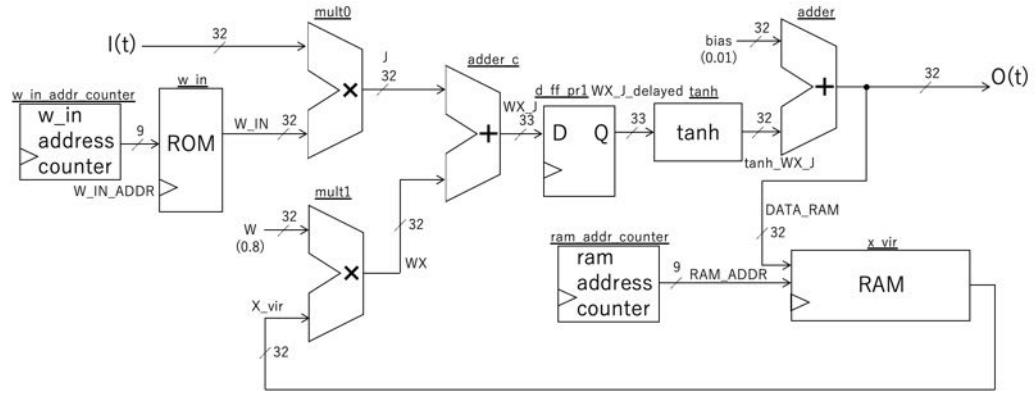


図 2.7 RAM の値変化と入力に対応した値

なお、本項以降では Ring Reservoir のアーキテクチャはこの RAM 型アーキテクチャを表すこととする。

前述の図 2.5 や図 2.6 のアーキテクチャでは、ノード数 $N = 3$ 、仮想ノード数 $N_{vir} = 4$ で構成しており、バイアス部も省略していた。これを、実用的なノード数（本論文では $N = 400, N_{vir} = 404$ ）に拡張し、バイアス項を組み込んだ。バイアス項は、演算毎に定数（本論文では 0.01）を加算することで実現した。実用的に拡張した RAM 型 Ring Reservoir のアーキテクチャを図 2.8 に示す。なお、図 2.6 で組み込んでいた wireJ のレジスタは、ROM をクロック同期型に変更したためクリティカルパスは解消したのと、このレジスタをなくすことによってそれぞれから加算までの遅延が同一になり、ROM と RAM のアドレスカウンタの開始タイミングを同期できることからレジスタは取り除いた。

また、仮想ノードに対する重み ($w = 0.8$) や、バイアス項に ($bias = 0.01$) の決め方については、後述するがここであらかじめ図に記入しておく。さらに、ビット幅についても同様に後述するがここでは 32bit としている。基本的にリザバーの入力・出力を含めた内部の値は -1 から 1 の間に収まり、小数点で扱うことができるところから、最上位 bit を符号 bit とし、上位第 2bit から小数点第一位としたため、ビット幅 32bit の場合 2 進数における小数点第 31 位まで扱うことを意味する。

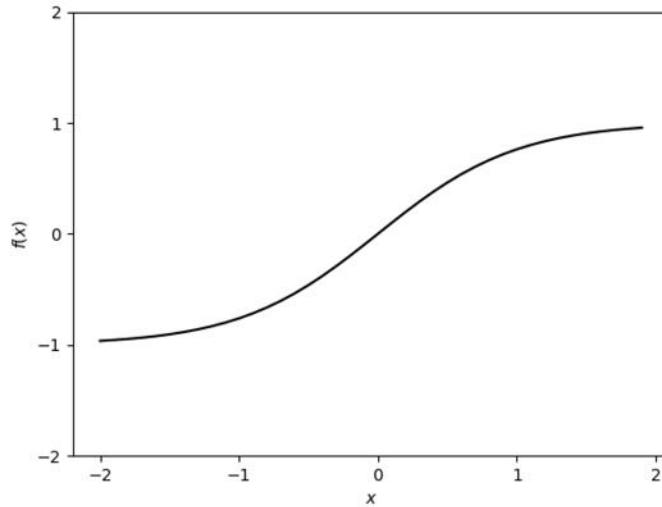
図 2.8 $N = 400$ に拡張した RAM 型 Ring Reservoir のアーキテクチャ

2.2.3 Piecewise Linear Function による省メモリ活性化関数

本項では、活性化関数について述べる。

活性化関数である \tanh をハードウェア上で正確に再現するためには LUT(Look Up Table) が必要になるが、その場合大規模なメモリが必要になる。ビット幅 32bit で構成する場合、LUT のワード数が約 43 億ワードと莫大な量になる。そこで、非線形な関数を区分的に線形な関数で置き換える PWL 関数 (Piecewise Linear Function, 区分線形関数) で代替することを考えた。

図 2.9 は $f(x) = \tanh(x)$ のグラフである。

図 2.9 \tanh 関数

この PWL として、 \tanh の 0 付近では線形で、正の値は 1 に負の値は -1 に漸近するという特徴を抽出して以下のような式を考えた。

$$f(x) = \begin{cases} -1 & (x < -1) \\ x & (-1 \leq x < 1) \\ 1 & (x \geq 1) \end{cases} \quad (2.7)$$

コレをグラフにすると図2.10 のようになる。

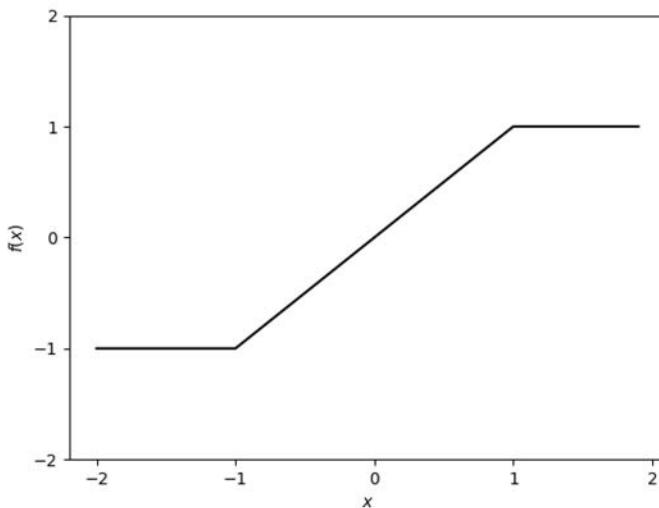


図 2.10 Piecewise Linear Function

このように、LUT では膨大な量のメモリが必要だったが、LUT に代替することで線形関数と条件分岐のみで構成することができるため回路規模が大幅に削減できる。

2.3 学習機との統合

2.3.1 シングルコア FORCE Learning Accelerator について

本研究では、考案した Ring Reservoir の FPGA アーキテクチャと先行研究によって実装済みである FORCE 学習の FPGA アーキテクチャを統合しひとつの FPGA 上に実装することで、FPGA 上で閉じた Reservoir Computing を実現することを試みた。なお、後述の高速 FORCE 学習機と区別するために、本項で用いる学習アーキテクチャはシングルコア FORCE Learning Accelerator と呼ぶ。

ここで、FORCE 学習における更新式を再掲する。

$$Z(t) = \vec{W}^T(t-1) \vec{R}(t) \quad (2.8)$$

$$e(t) = Z(t) - S(t) \quad (2.9)$$

$$\hat{P}(t) = \hat{P}(t-1) - \frac{\hat{P}(t-1) \vec{R}(t) \vec{R}^T(t) \hat{P}(t-1)}{1 + \vec{R}^T(t) \hat{P}(t-1) \vec{R}(t)} \quad (2.10)$$

$$\vec{W}(t) = \vec{W}(t-1) - e(t) \hat{P}(t) \vec{R}(t) \quad (2.11)$$

先行研究（第1章 [8]）で構築された FORCE 学習アーキテクチャを図 2.11 に示す。

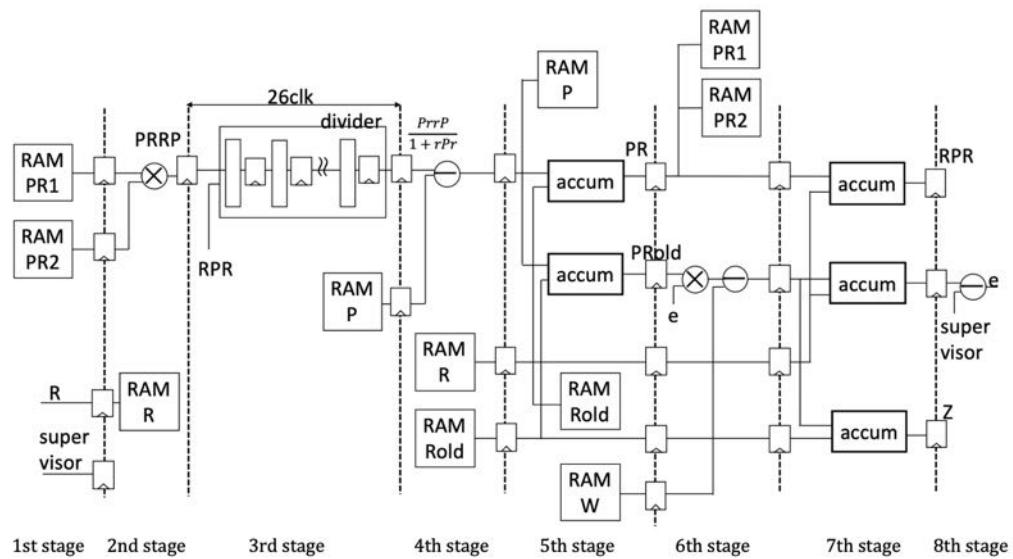


図 2.11 シングルコア FORCE Learning Accelerator のアーキテクチャ

以下に、アーキテクチャの動作フローを示す。

第1ステージでは、前サイクルに計算された $\hat{P}(t-1) \vec{R}(t)$ と $\vec{R}^T(t) \hat{P}(t-1)$ 、入力 R と教師信号の読み出しを行う。 $\hat{P}(t)$ は対称行列であるため、 $\hat{P}(t-1) \vec{R}(t)$ と $\vec{R}^T(t) \hat{P}(t-1)$ は同一のベクトルであるが、同時に別の読み出しアドレスを指定する必要があるためそれぞれ2つのRAMに保存されている。第2ステージでは2つのRAMから読み出されたデータ同士の乗算を行う。また、 R はRAMに格納される。第3ステージでは、第2ステージの演算結果を前サイクルに計算された $1 - \vec{R}^T(t) \hat{P}(t-1) \vec{R}(t)$ で割る。なお、途中で1から減算する計算も行っている。ここでは除算アルゴリズムである復元法を用いて、被除数桁回数の除算を行う。そのため、このステージでは被除数桁回数分のCLKがかかる。第4ステージでは、RAMから読み出された $\hat{P}(t-1)$ と第3ステージの演算結果の差をとる。RAMはアドレス0から順にカウントアップされていく、 $N^2 - 1$

までを読み出す。この RAM は N^2 ワード必要であるため、外部 SRAM を用いている。第 5 ステージでは、まず更新された $\hat{P}(t)$ を RAM に書き戻す。ただし、外部 SRAM は 1 ポート SRAM を 2 つ使用するため、読み出しに使った SRAM とは別の SRAM に書き込みを行う。次のサイクルでは、書き込んだ SRAM から読み出しを行い、もう一方の SRAM に書き込む。また、更新された $\hat{P}(t)$ と $\vec{R}(t)$ 、 $\vec{R}(t+1)$ との積和演算を行うことで NCLK かけて $\hat{P}(t)\vec{R}(t)$ と $\hat{P}(t)\vec{R}(t+1)$ を計算する。 $\hat{P}(t)\vec{R}(t+1)$ は次のサイクルに $\hat{P}(t-1)\vec{R}(t)$ となり、第 1 ステージで利用される。なお、この際の $\vec{R}(t+1)$ は第 2 ステージで RAM に格納された入力であり、 $\vec{R}(t)$ が前のサイクルで受け取った R である。つまり、最新の入力は $\vec{R}(t+1)$ として受け取り、次サイクルではそれを $\vec{R}(t)$ として計算に用いている。第 6 ステージでは、重みの更新を行う。 $\hat{P}(t)\vec{R}(t)$ は NCLK ごとに計算結果が現れるため、同様に NCLK かけて 1 つの重みを更新する。第 7 ステージでは、 $\hat{P}(t)\vec{R}(t+1)$ と $\vec{R}(t+1)$ の積和演算を行うことで $\vec{R}^T(t+1)\hat{P}(t)\vec{R}(t+1)$ を、 $\vec{W}(t)$ と $\vec{R}(t)$ の積和演算を行うことで $Z(t)$ を計算する。 $\vec{R}^T(t+1)\hat{P}(t)\vec{R}(t+1)$ は次のサイクルで $\vec{R}^T(t)\hat{P}(t-1)\vec{R}(t)$ として第 3 ステージの除算部分で用いられ、 $Z(t)$ は予測結果として出力される。最後の第 8 ステージでは、次サイクルの第 6 ステージで使用される教師信号と出力値との差 e を求める。

なお、先行研究での検証により、400 ノード、符号付固定小数点 26bit が NARMA10 タスク等に対して妥当であることが示されており、本研究においても同じ条件で用いる。

2.3.2 統合

前述の Ring Reservoir とシングルコア FORCE Learning Accelerator を結合し、1 つの FPGA で入力信号から学習・推論まで完結する 400 ノード-Reservoir Computer のアーキテクチャを実現した。全体の FPGA アーキテクチャの概略図を図 2.12 に示す。

このアーキテクチャを CycloneIV EP4CE10F17C8 に実装した。評価環境としては、PC と接続した Arduino と評価用 FPGA ボードをシリアル通信で接続し、FPGA に入力信号、教師信号を与える、学習・推論した出力値を取得した。評価用 FPGA ボード、その動作環境と主な構成要素をそれぞれ図 2.13 と表 2.1 に示す。

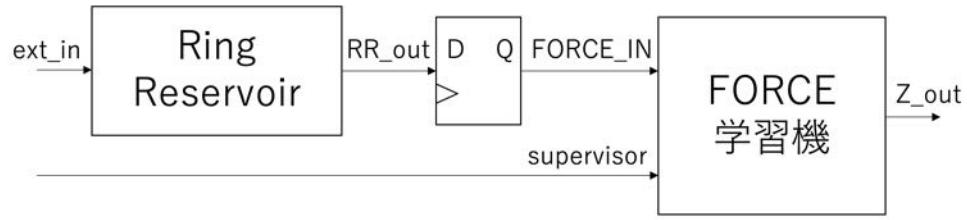
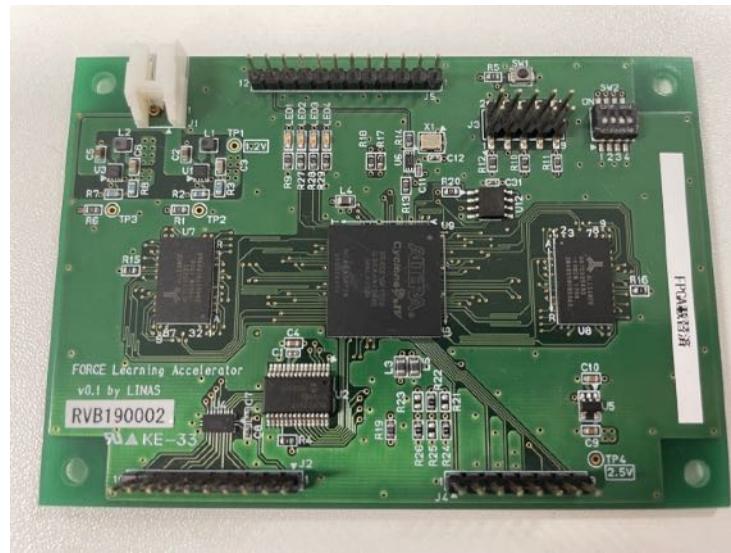


図 2.12 全体アーキテクチャ



2.4 評価と考察

2.4.1 Ring Reservoir アーキテクチャの評価

まず、Ring Reservoir 単体での評価を述べる。リザバー部の評価であるため、学習は一般的な手法である線形回帰を PC 上で行った。また、図 2.19 までの評価における活性化関数は、PWL との比較のために LUT で行っている。

はじめに、bit 制約がない場合のシミュレーション評価結果 (NARMA10・MC) を図 2.14 と図 2.15 に示す。また、評価指標は以下のようになつた。

$$NRMSE = 0.0269 \quad (2.12)$$

$$MC = 53.14 \quad (2.13)$$

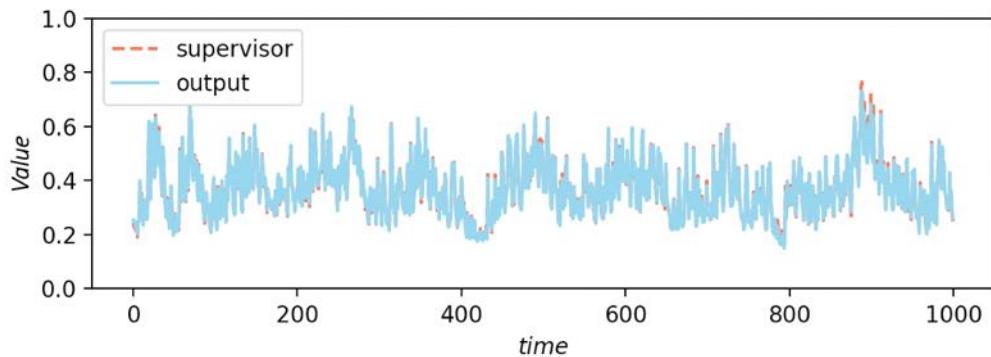


図 2.14 NARMA10 タスクに対する精度 (bit 制約なし)

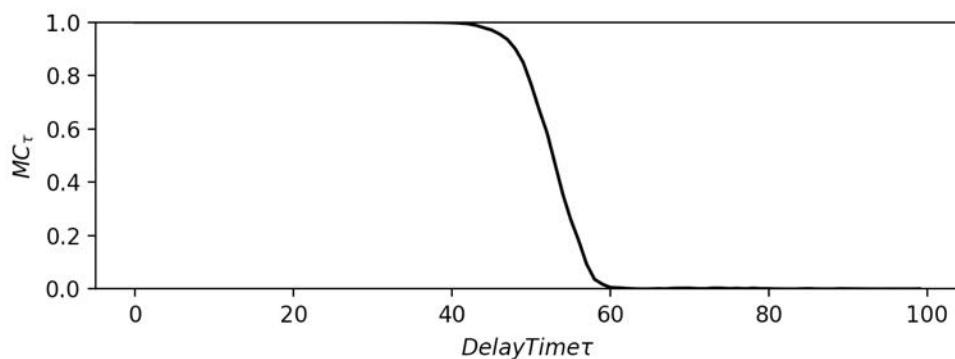


図 2.15 MC 測定による記憶容量評価 (bit 制約なし)

参考として、ESN をシミュレーション上で同様に評価した結果を図 2.16 と図 2.17 に示す。なお、内部結合重みはスペクトル半径 0.9 として事前調整している。

NARMA10 タスクに対する NRMSE と MC は以下の通りである。

$$NRMSE = 0.03606 \quad (2.14)$$

$$MC = 45.49 \quad (2.15)$$

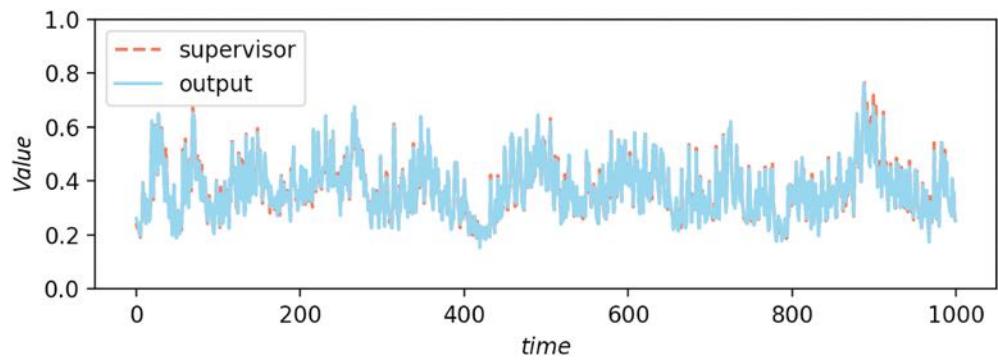


図 2.16 NARMA10 タスクに対する精度 (ESN)

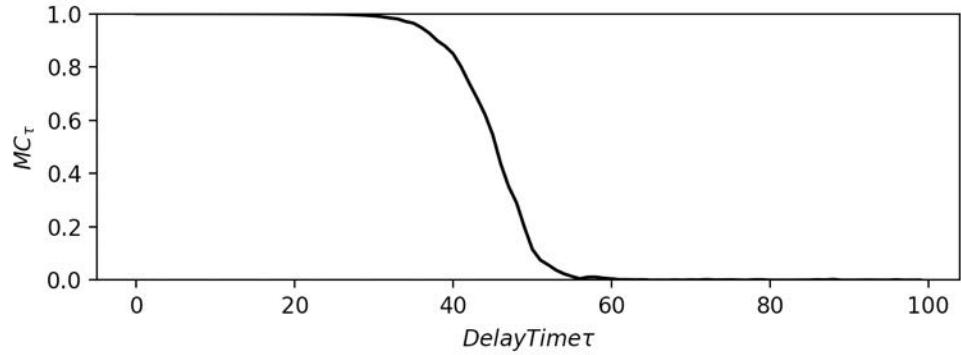


図 2.17 MC 測定による記憶容量評価 (ESN)

Ring Reservoir と ESN の評価結果を比較すると、精度・容量どちらについても十分な処理能力があるということがわかる。なお、NRMSE、MC とともに Ring Reservoir の方が良い結果を出しているが、これは一概に Ring Reservoir の方が性能が良いということではない。Ring Reservoir は今回の検証用に最適なパラメータに調整しているが、本研究の主題ではない ESN にはスペクトル半径以外のパラメータ調整を行っていない。

また、内部重み w やバイアス項の値については、この精度や記憶容量を元に $w = 0.8, bias = 0.01$ が最適だと分かった。

続いて、シミュレーション上で本論文の考案アーキテクチャである 32bit に制約をかけた場合の評価結果を図 2.18 と図 2.19 に示す。また、評価指標は以下の通りである。

$$NRMSE = 0.02692 \quad (2.16)$$

$$MC = 53.14 \quad (2.17)$$

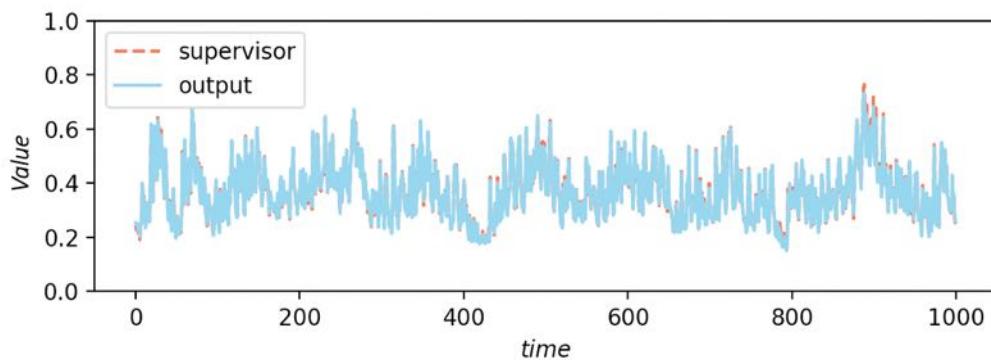


図 2.18 NARMA10 タスクに対する精度 (32bit)

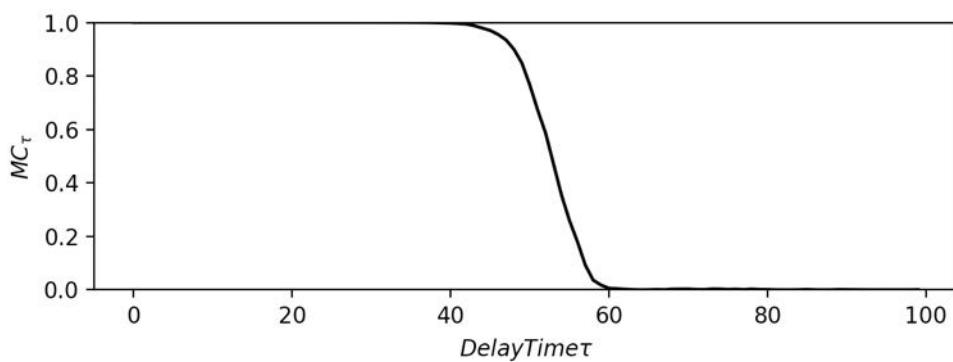


図 2.19 MC 測定による記憶容量評価 (32bit)

結果は、bit 制約をかけていない結果とほぼ一致し、32bit 制約では ESN に近い精度・記憶容量で動作することがわかる。さらに制約 bit 幅を下げていくと、性能が低下していくことが追加検証したシミュレーションで分かった。wire ごとに適切な bit 幅を探索していくことでアーキテクチャ全体として最適な bit 幅を選定することは可能である

が、本研究の目的は実用化であるため、その bit 幅の探索は行わず FPGA 実装に用いるアーキテクチャは 32bit のものにすると決定した。

次に、Ring Reservoir の FPGA アーキテクチャを Intel Cyclone 10 LP (型番:10CL016YM164) に実装した際のハードウェアリソースを示す。なお、前述の通り 32bit で構成しており、活性化関数については PWL を用いている。

Flow Summary	
<<Filter>>	
Flow Status	Successful - Fri Nov 27 12:50:03 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Standard Edition
Revision Name	RingReservoir_cyc10
Top-level Entity Name	RingReservoir_cyc10
Family	Cyclone 10 LP
Device	10CL016YM164I7G
Timing Models	Final
Total logic elements	311 / 15,408 (2 %)
Total registers	51
Total pins	66 / 88 (75 %)
Total virtual pins	0
Total memory bits	32,768 / 516,096 (6 %)
Embedded Multiplier 9-bit elements	16 / 112 (14 %)
Total PLLs	0 / 4 (0 %)

図 2.20 compile summary

この summary を見ると、Total logic elements は 300 程で FPGA(Cyclone 10) の全体の 2%にしか及ばないことがわかる。Reservoir Computing として FPGA 上で完結させる場合、学習機を乗せる必要があるが、この程度の非常にコンパクトな回路であれば、十分に乗せることができると考えられる。実際に学習機と統合した際の評価については後述する。

2.4.2 Piecewise Linear Function による性能変化

次に、PLW 代替による性能変化について示す。

活性化関数を PWL に置き換えた場合の評価結果を図 2.21 と図 2.22 に示す。また、評価指標を以下の表にまとめた。なお、前述の通り 32bit で構成している。

表 2.2 LUT と PWL の比較

	NRMSE	MC
LUT	0.02692	53.14
PWL	0.1088	81.01

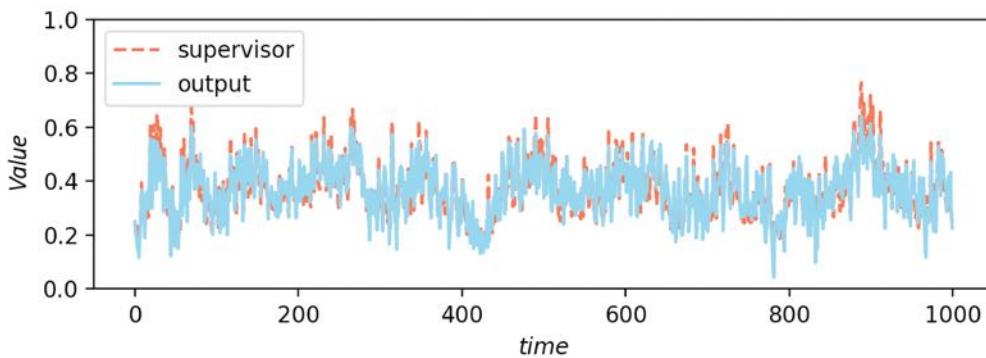


図 2.21 NARMA10 タスクに対する精度 (32bit, PWL)

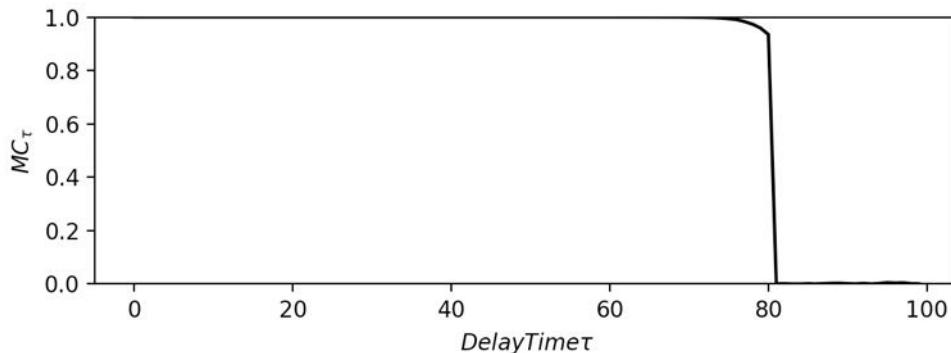


図 2.22 MC 測定による記憶容量評価 (32bit, PWL)

結果から、NARMA10 タスクに対する精度は低下したが記憶容量は増加したことがわかる。これは、活性化関数の非線形形成分の大部分が線形形成分に置き換わったことによる影響だと考えられる。NARMA10 は非線形タスクであるため、活性化関数を区分的な線形関数に代替することで、NARMA10 の非線形性を再現しづらくなり精度が低下したと考えられる。また、MC は線形容量とも呼ばれ、リザバーの線形的な記憶容量を測る評価指標である。そのため、PWL 代替によって増加したと考えられる。

また、ここまで結果は PWL の線形部の傾きが 1 であったが、傾きを図 2.23 のように変更することで、異なる特性が見つかる可能性があると考え、傾きを変更して追

加検証を行った。その NRMSE と MC を表 2.3 にまとめた。

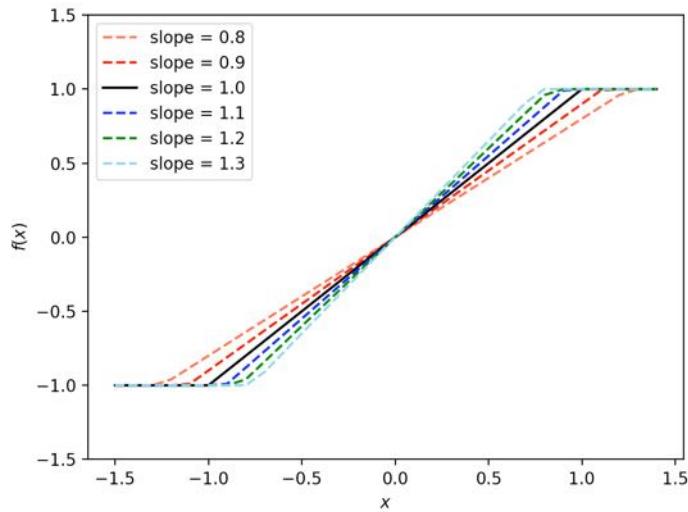


図 2.23 PWL の傾き

表 2.3 PWL の傾きを変えたときの NRMSE と MC の変化

傾き	NRMSE	MC
0.8	0.1074	43.64
0.9	0.1091	58.87
1	0.1088	81.01
1.1	0.1104	81.18
1.2	0.1113	81.04
1.3	0.2871	0.3231

この結果から、傾きが 0.8 から 1.2 までは NRMSE にあまり大きな変化は見られないことがわかる。一方で、傾き 1.3 の場合精度が大きく落ちている。なぜ傾き 1.3 から大きく精度が落ちたのか考える。傾き 1.3 の場合、PWL への入力の絶対値が約 0.769 で PWL 出力が 1 または -1 になる。本研究では内部重みを 0.8、入力結合重みを 0.01 としているため、一度出力が 1 または -1 になってしまふとその後内部重みで 0.8 をかけられ、PWL 出力が再び 1 または -1 になってしまふ可能性が高い。つまり、一度 PWL の上限または下限に触れたノード値は以降のサイクルでもそのままの可能性が高くなる。これが原因で精度が大きく低下していると考えられる。

一方、MC に着目すると、傾きが 0.9 以下では MC が低下していることがわかる。上でも述べたが、内部重みを 0.8、入力結合重みを 0.01 としているため、傾きが 1 より小

さい場合、リング上を回るたびに値が小さくなっていくだけになり、古い情報は下位ビットに流れていき 32bit という制約で比較的速く忘却してしまうからであると考えられる。

次に、IPC の測定結果を図 2.25 に示す。なお、活性化関数について LUT と PWL に加え、比較用に PWL よりもさらに tanh に近い、式 2.18 で表される 2 次近似関数(図 2.24)を定義し、これも含め 3 種類の活性化関数で IPC を評価した。

$$f(x) = \begin{cases} -1 & (x < -1) \\ 1 - (1 - x)^2 & (-1 \leq x < 0) \\ (1 + x)^2 - 1 & (0 \leq x < 1) \\ 1 & (x \geq 1) \end{cases} \quad (2.18)$$

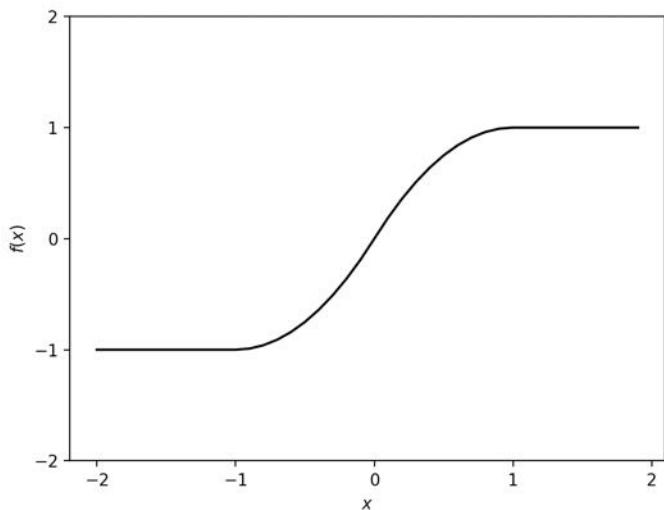


図 2.24 2 次近似関数

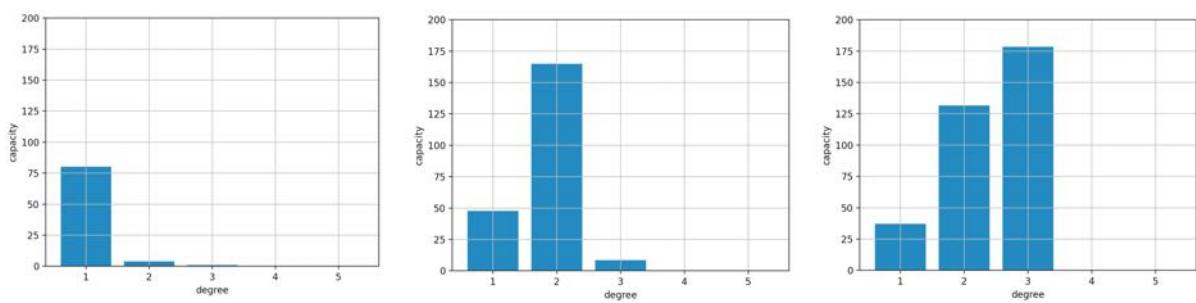


図 2.25 PWL(左)、2 次近似関数(中)、LUT の IPC 測定結果

図 2.25 の結果から、PWL、2 次近似関数、LUT の順で高次側に capacity がシフトしていることがわかる。LUT は \tanh を可能な限り再現しているので、ティラー展開をすると次数が無限大の関数だと考えることができ、PWL、2 次近似関数、LUT の順で最大次数が大きい関数であると置き換えることができる。また、各次数に対する capacity の大きさは教師信号における各次数に対する説明能力の大きさと捉えることができる。したがって、活性化関数の高次性を高めることにより高次の教師信号に対する説明能力が向上すると考えることができる。また、1 次における capacity は線形タスクに対する記憶容量であり MC と同等である。MC と非線形性にはトレードオフの関係があることも知られており、関数の高次性が増すほど 1 次成分が小さくなっていることはそのことからも説明できる。

なお、全ての条件において 4 次以降の capacity が存在しないのは、量子化誤差 (32bit) を考慮したシミュレーションであるのが原因であると推測される。一般的なティラー展開を考えると、高次項になるほど係数は小さくなる。よって、レザバー内で高次成分は比較的下位ビットに記憶されると考えることができ、量子化を行った場合その高次成分の説明能力を失うため、今回の測定では 4 次以降の capacity が消失したと考えられる。

2.4.3 学習機との統合評価

最後に、Ring Reservoir とシングルコア FORCE Learning Accelerator を結合したアーキテクチャにおける NARMA10 タスクに対する学習結果の一部を図 2.26 に、そのときの NRMSE を以下に示す。なお、アーキテクチャ中の活性化関数は PWL を用いている。

$$NRMSE = 0.159 \quad (2.19)$$

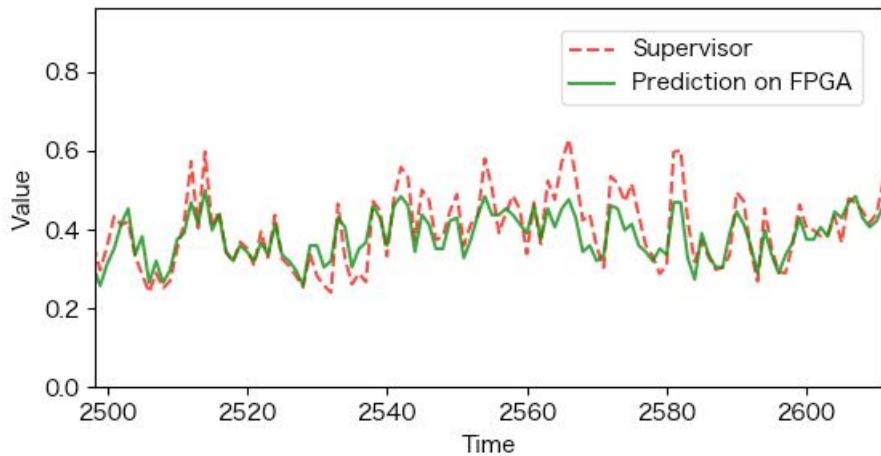


図 2.26 学習機と統合後の NARMA10 タスクに対する精度

NRMSE は 0.159 となり、線形回帰による学習でのシミュレーション結果である 0.1088 より精度は低下したが、一般にオンライン学習である FORCE 学習の方が精度は低下する。数値的に厳密な精度を必要としない用途であれば、これを 1 つの FPGA で完結した Reservoir Computer として様々な用途に応用することができると考えられる。

付録A FPGA Reservoir Computerの動作手順

2章で構築したFPGA Reservoir Computerの動作手順について示す。

デバイスの接続

下図に従ってデバイスを接続する。

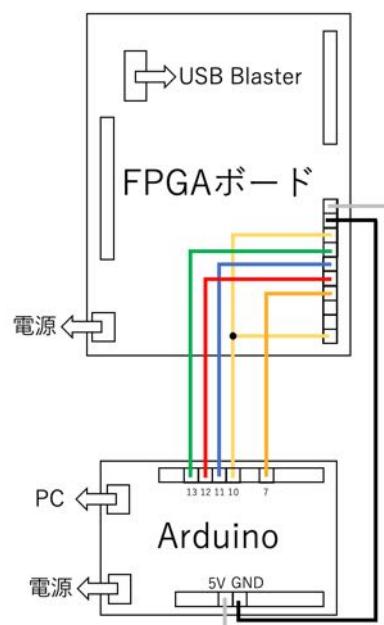


図 A.1 デバイス間の配線

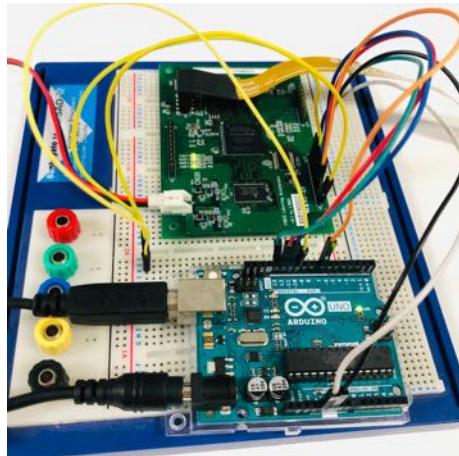


図 A.2 接続時の写真

FPGAへの書き込み

QuartusでFPGAにsofファイル(/RingReservoirFORCE/output_files/FORCE_Accelerater.sof)を書き込む。

Arduinoへの書き込み

Arduino IDEでinoファイル(/Arduino/reservior_spi/reservior_spi.ino)を書き込む。

(注) ライブラリ「MCP23S17」必須

(スケッチ>ライブラリをインクルード>.ZIP形式のライブラリをインストール
>/Arduino/MCP23S17.zipを選択)

Pythonの実行

実行用のPythonコード(/Python/Serial_test.py)を実行。

(注) 外部ライブラリ：numpy, matplotlib, pyserial 必須

インストール用コマンド：

```
$ pip install numpy  
$ pip install matplotlib  
$ pip install pyserial
```

```
(FORCE) k_yoshida@k-yoshidamacbook:Python$ python3 Serial_test.py
Open Port
Initialize Arduino
time: 0
ext_in:0.273
supervisor:0.42
prediction:1.0
time: 1
ext_in:0.352
supervisor:0.5
prediction:0.391
time: 2
ext_in:0.297
supervisor:0.576
prediction:0.461
time: 3
ext_in:0.266
supervisor:0.477
prediction:0.523
```

図 A.3 動作中の画面

学習と推論の切り替え方法

学習と推論は SW2-1 番で切り替え可能。

OFF : 学習

ON : 推論

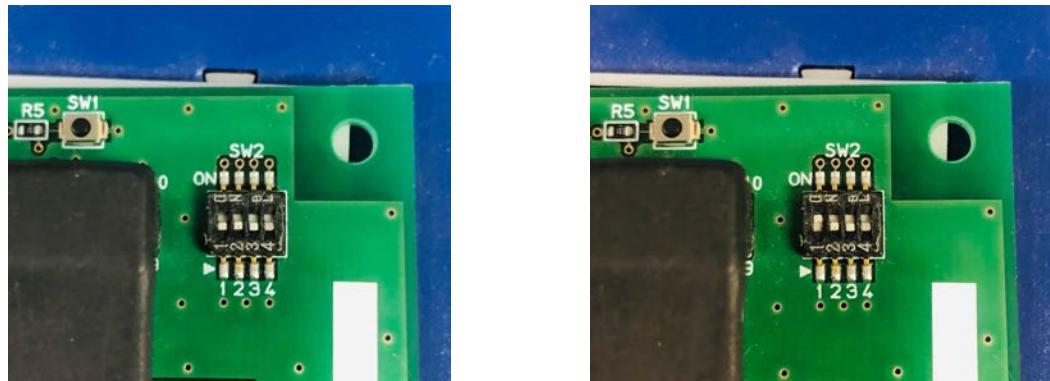


図 A.4 学習時 (左) と推論時 (右) の SW2 の写真

プログラム終了

終了時刻 (デフォルト:1000) に達する、または KeyboardInterrupt でプログラムが終了し、教師信号と予測結果をグラフに表示。

予測結果と教師信号は npy ファイル (np_save_o.npy, np_save_s.npy) へ出力。

(注) 学習と推論の切り替えのタイミングで予測値の読み込みが 1 分スキップされるため、予測値のデータ長は教師信号のデータ長-1 になる。

(注) KeyboardInterrupt で強制終了した場合、タイミングによっては予測結果未取得で終了することがある。その場合、さらに予測値のデータ長は-1 になる。

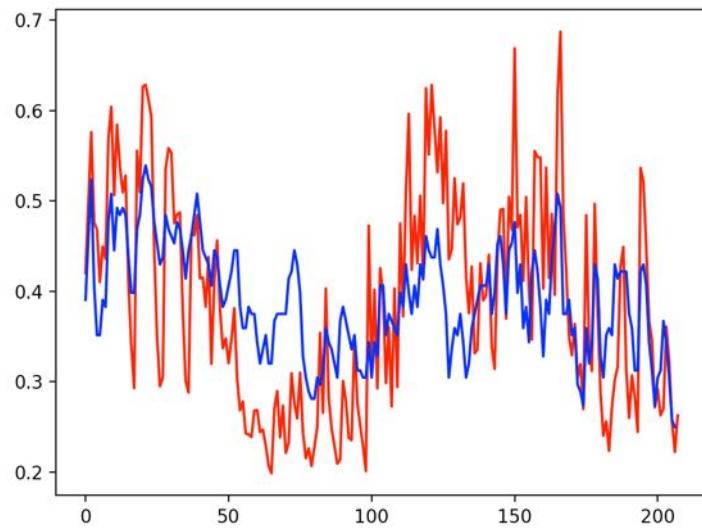


図 A.5 終了後の画面

予測結果評価

評価用 Python コード (/Python/FORCE_Evaluation.py) を実行して、予測結果の再表示、エラーの評価が可能。

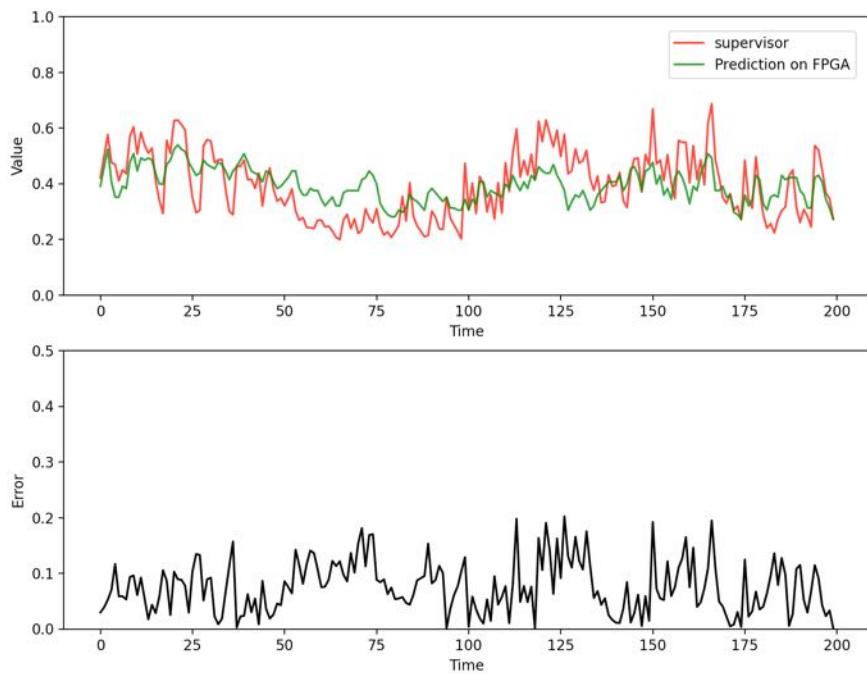


図 A.6 教師信号と予測値の推移(上)とそのときの二乗誤差(下)

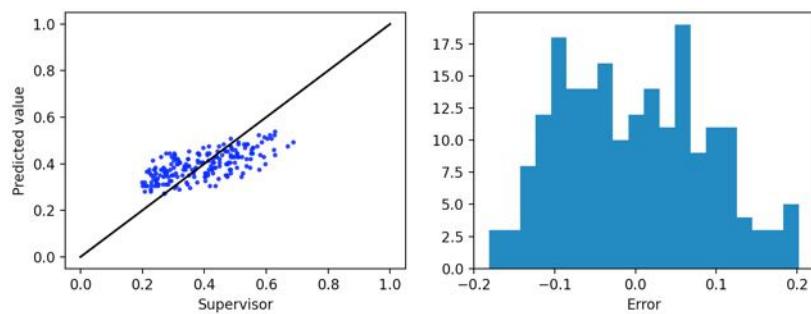


図 A.7 教師信号と予測値の散布図(左)と誤差ヒストグラム(右)

