C-021

論理ゲート変換によるニューラルネットワークのディジタル回路化
Digital Circuit Implementation of Neural Network Based on Logic Gate Conversion

立石直宏 † 重井徳貴 † 寺村正広 ‡ 宮島廣美 †
Naohiro Tateishi Noritaka Shigei Masahiro Teramura Hiromi Miyajima

1. まえがき

ニューラルネットワーク (NN) のアナログ回路化 [1] やディジタル回路化 [2, 3] について検討がなされている．ディジタル回路化に関しては, Dinu らが, 乗算器を使用せずゲート回路に基づく NN の構成法を提案している [3]．この方法は, 原理的に丸め誤差が発生せず, 入力のビット数が少ない場合, 乗算器を用いるものよりも回路を小さくできる．一方, FPGA より安価な CPLD は,AND-OR 構造の回路を基本単位としており, 乗算器を使用するとリソースが逼迫してしまう．

そこで本研究では, CPLD などの安価なデバイスに実装することを想定し, 回路面積と精度の面で効果的な手法を提案する．提案する手法は, 乗算器を使用せず, ゲート変換により NN を実現する．提案法は, 面積と精度のトレードオフを考慮したゲート変換を行い, 回路面積の面で Dinu らの手法よりも優れ, 精度の面で乗算器を用いる手法よりも優れることを示す．

2. ニューロンモデルのディジタル化

本研究で対象とするアナログのニューロンモデルを図 1 に示す．x_m は入力, w_m は重み, θ は閾値であり, 出力 y は次式で与えられる．

$$y = f\left(\sum_{i=1}^{m} w_i \cdot x_i - \theta\right) = f(u) \qquad (1)$$

ただし, $u \geq 0$ のとき $f(u) = 1$, それ以外は $f(u) = 0$ である．このニューロンモデルをディジタル回路化する場合, 式 (1) 中の $\sum_{i=1}^{m} w_i \cdot x_i$ を求めるときに, 通常, 乗算器を使用する．

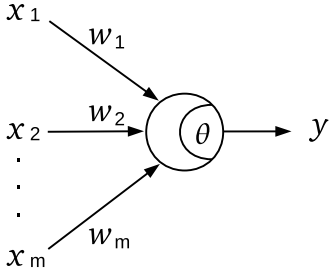

図 1: ニューロンモデル

本研究ではニューロンモデルを AND ゲートと OR ゲートのみの論理ゲート構造に変換するために, 式 (1) のニューロンモデルの入力を”0” か ”1” のバイナリ表示にディジタル化する．

†鹿児島大学大学院理工学研究科
‡川内職業能力開発短期大学校

入力 x_i を $x_{i,n-1}\ x_{i,n-2} \ldots x_{i,p} \ldots x_{i,1}\ x_{i,0}$ のように, n bit のバイナリ表示に変換した場合, x_i に対応する重み w_i は, 各 bit 毎に分配することができる．n bit に変換した x_i の p bit 目, $x_{i,p}$ に対応する重み $w_{i,p}$ は以下のように計算される．

$$w_{i,p} = \frac{2^p}{2^n} \cdot w_i \qquad (2)$$

入力がディジタル化されたニューロンモデルを図 2 に示す．m 個の入力はそれぞれ n bit に変換されており, 入力に対応する重みが, 各 bit 毎に分配されている．

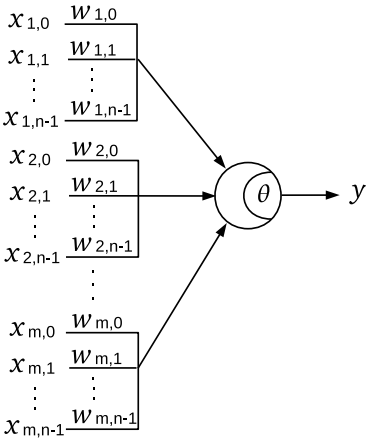

図 2: 入力がディジタル化されたニューロンモデル

入力はバイナリ表示で “0” と “1” のみとなる $x_{1,0}$ 〜 $x_{m,n-1}$, 重みは $w_{1,0}$ 〜 $w_{m,n-1}$, 閾値 θ は図 1 の θ と同じ値となり, 出力 y は次式で与えられる．

$$y = f\left(\sum_{i=1}^{m}\sum_{p=0}^{n-1} w_{ip} \cdot x_{ip} - \theta\right) = f(u) \qquad (3)$$

3. ニューロンモデルの論理ゲート構造への変換

Dinu らは, 入力がディジタル化されたニューロンモデルを論理ゲート構造からなる回路に変換するために, 重みの情報に基づく変換アルゴリズムを提案している [3]．素朴な方法として, 真理値表を用いる方法が考えられるが, これは入力のビット数が多くなると変換回路を小さくすることが困難である．以下では Dinu らのアルゴリズムについて説明する．

3.1. 重みの情報に基づく変換

重みの情報に基づくニューロンモデルの論理ゲート構造への変換には, 入力重み w_j^{s} と累積重み w_i^{t} を用いる．

- 入力重み w_j^{s}
  入力 x_1〜x_m の n bit バイナリ表示に対応する重み $w_{1,0} \sim w_{m,n-1}$ を降順に並べ替えたとき, w_j^{s} は $j \in \{1, 2, \ldots, m \cdot n\}$ 番目に大きい重みである.

- 累積重み w_i^{t}
  w_j^{s} における $j = i$ から $j = m \cdot n$ までの累積 $w_i^{\mathrm{t}} = \sum_{j=i}^{m \cdot n} w_j^{\mathrm{s}}$ である.

3.2. 論理ゲート構造への変換アルゴリズム

論理ゲート構造への変換アルゴリズムを以下に示す. Step3 “AND ゲートの作成” にて, θ が w_j^{s} によって更新されることでゲートが生成されていく.

Step1　最初に回路に追加する論理ゲートの種類の判断
追加される論理ゲートは, $j \in \{1, 2, ..., m \cdot n\}$ に対応する入力 $x_{i,p}$ を含んでいる.

(1.1) $w_j^t > \theta$ を満たす w_j^t が複数個あれば, OR ゲートとなる. $jc = 1$ として, Step2 へ.

(1.2) それ以外では AND ゲートとなる. $jc = 1$ として, Step3 へ.

Step2　OR ゲートの生成
次の手続き add_OR(jc, θ) を呼び出す.

Procedure add_OR(jc, θ)
　入力：jc, θ;
begin
　$j \leftarrow jc$;
(2.1)　**while** $w_j^{\mathrm{s}} > \theta$ **do**
　w_j^{s} に対応する入力 $x_{i,p}$ を OR の入力に追加;
　$j \leftarrow jc$;
(2.2)　**while** $w_j^{\mathrm{t}} > \theta$ **do**
　add_AND(j, θ) を呼び出し, AND を生成;
　この AND の出力を OR の入力に追加;
　$j \leftarrow jc$;
end

Step3　AND ゲートの生成
次の手続き add_AND(jc, θ) を呼び出す.

Procedure add_AND(jc, θ)
　入力：jc, θ;
begin
　$j \leftarrow jc$;
(3.1)　$\theta \leftarrow \theta - w_j^{\mathrm{t}}$;
　w_j^{s} に対応する入力 $x_{i,p}$ を AND の入力に追加;
　$j \leftarrow j + 1$;
(3.2)　**while** $w_j^{\mathrm{s}} < \theta$ **and** $w_{j+1}^{\mathrm{t}} < \theta$ **do**
　$\theta \leftarrow \theta - w_j^{\mathrm{s}}$;
　w_j^{t} に対応する入力 $x_{i,p}$ を AND の入力に追加;
　$j \leftarrow j + 1$;
(3.3)　add_OR(j, θ) を呼び出し, OR を生成;
　この OR の出力を AND の入力に追加;
end

(アルゴリズム終了)

手続き add_AND(j, θ) の (3.3) では, 再帰的に自身を呼び出している. 再帰が深くなるにつれ, 引数 θ の値は小さくなる. これは関連するゲートを生成しなくても誤差は小さいことを意味する. そこで 4 で説明する提案法では, 手続き add_AND(j, θ) の (3.3) を変更することで, 論理ゲート構造に変換する際に使用するゲート数の削減を行う.

3.3. 論理ゲート構造への変換例

2 入力 1 出力のニューロンモデルにおいて, 入力を 4 bit とした場合の論理ゲート構造への変換例を示す. 入力は x_1, x_2, 重みは w_1, w_2, 閾値は θ であり, それぞれの値は $0 \leq x_1, x_2 \leq 1$, $w_1 = 0.36$, $w_2 = 1.00$, $\theta = 0.43$ とする. 論理ゲート構造の変換に用いた w_j^s, w_j^t の値を表 1 に示す. 論理ゲート構造への変換アルゴリズムにより作成された回路を図 3 に示す. 図 3 に示す回路は, 出力 y から深さ優先探索の順番にゲートが生成され, OR ゲート と AND ゲートを繰り返した構造となる.

表 1: w_j^{s}, w_j^{t} の値

j	w_j^{s}	w_j^{t}	$x_{i,p}$
1	0.5000	1.2750	$x_{2,3}$
2	0.2500	0.7750	$x_{2,2}$
3	0.1800	0.5250	$x_{1,3}$
4	0.1250	0.3450	$x_{2,1}$
5	0.0900	0.2200	$x_{1,2}$
6	0.0625	0.1300	$x_{2,0}$
7	0.0450	0.0675	$x_{1,1}$
8	0.0225	0.0225	$x_{1,0}$

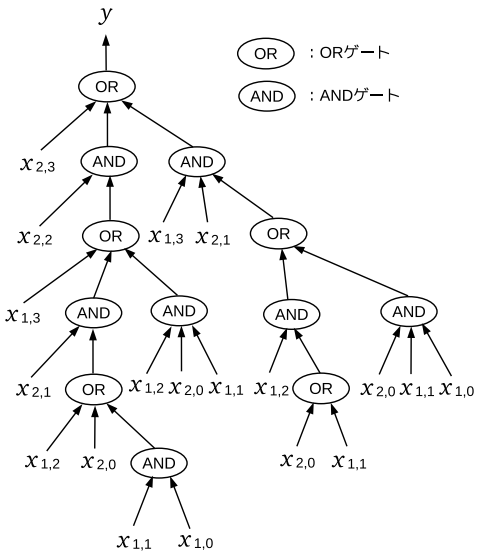

図 3: ニューロンモデルの論理ゲート構造への変換例

4. 提案法

提案法では, ニューロンモデルの論理ゲート構造に変換する際に使用するゲート数を削減することで, 回路面積の縮小を行う. ニューロンモデルの論理ゲート構造への変換は, θ を w_j^{s} によって更新しながらゲートを生成していくことで行われる. よって θ の更新に用いる

w_j^{s} の個数を減らすことでゲート数を削減できる. θ の更新は, 論理ゲート構造への変換アルゴリズム Step3, “AND ゲートの作成” にて行われており, Step3 の終了条件を変更することで, θ の更新に用いる w_j^{s} の個数を減らす.

4.1. ゲート数削減のアルゴリズム

提案法のアルゴリズムは 3.2 で述べたものと Step3 の (3.3) の部分が異なる.

提案アルゴリズムでは, パラメータ $\eta \in \{1, \ldots, m \cdot n - 1\}$ により論理ゲート変換に使用するゲート数を調整する. η は θ の更新に用いない w_j^{s} の個数のことであり, η が大きくなるほど論理ゲート変換に使用するゲート数は少なくなる. 提案アルゴリズムの Step3 の (3.3) を以下に示す.

Step3

(3.3) $w_j^{\mathrm{t}} > \theta$ を満たす w_j^{t} の個数 k をカウント;
  if $1 \leq k \leq \eta + 1$
    w_j^{s} に対応する入力 $x_{i,p}$ を AND の入力に追加;
    論理ゲート構造への変換終了;
  else
    add_OR(j, θ) を呼び出し, OR を生成;
(Step3 終了)

4.2. 提案法による論理ゲート構造の変換例

表 1 の w_j^{s}, w_j^{t} を用いて, $\eta = 3$ とした場合に, 提案法により作成された回路を図 4 に示す.

図 3 の回路と比較すると, 回路の末端のゲートが削除されてることがわかる. 図 3 の回路は, 表 1 の $j = 1$〜7 に対応する w_j^{s} を用いて θ の更新を行っているのに対して, 図 4 の回路は表 1 の $j = 1$〜4 に対応する w_j^{s} を用いて θ の更新を行っている.

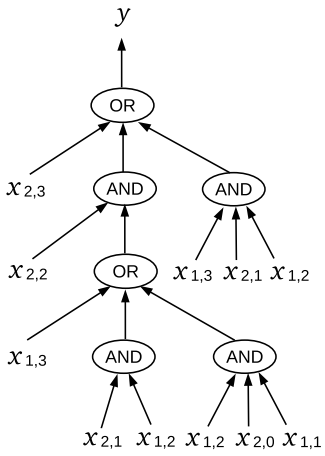

図 4: 提案法による論理ゲート構造への変換例

5. 評価

2 入力 1 出力のニューロンモデルにおいて, 入力を 8 bit と 10 bit にバイナリ変換し, 以下の回路についてシミュレータ上での出力誤差と回路面積を比較する.

- 乗算器を使用して作成した回路
  (乗算器は, 回路面積を抑えるため, 乗算結果の下位 bit 半分を切り捨てる.)
- Dinu らの論理ゲート変換で作成した回路
- 提案法の論理ゲート変換で作成した回路

回路の設計環境として言語に VHDL, 設計ソフトウェアに Xilinx ISE 10.1i, 対象デバイスに Xilinx 社 CoolRunner-II を用いた. 論理ゲート変換の場合は, 変換アルゴリズムにより生成されたゲート構造を VHDL で記述し, 設計ツールで論理合成した.

評価に用いたニューロンモデルの理想的な出力境界線を図 5 に示す. 入力 x_1, x_2 が図 5 に示す出力境界線の上側にある場合, 出力 y は”1” を出力し, 下側にある場合, 出力 y は”0” を出力する. 入力 8 bit での結果を表 2, 入力 10 bit での結果を表 3 に示す. 表 2, 表 3 における各項目は以下の通りである.

- η
  提案法において, θ の更新に用いない重みの個数
- ゲート
  変換アルゴリズムが生成した AND ゲートと OR ゲートの総数
- P Term
  CPLD 上で使用された積項 (Product Term) の数
- Macrocell
  CPLD 上で使用された Macrocell の数
- 誤り率
  作成回路が出力を誤った割合 (倍精度の浮動小数点数で求めたものを正しい出力とした)

表 2: 入力 8 bit での比較結果

		ゲート	P Term	Macrocell	誤り率 [%]
乗算器		—	228	50	0.177
Dinu らの手法		334	97	14	0
提案法	$\eta = 1$	274	94	17	0.021
	$\eta = 2$	240	89	22	0.090
	$\eta = 3$	193	66	11	0.174
	$\eta = 4$	152	62	9	0.395
	$\eta = 5$	115	50	6	0.638

表 3: 入力 10 bit での比較結果

		ゲート	P Term	Macrocell	誤り率 [%]
乗算器		—	418	89	0.091
Dinu らの手法		1356	372	46	0
提案法	$\eta = 1$	1107	345	42	0.006
	$\eta = 2$	954	325	40	0.023
	$\eta = 3$	801	273	40	0.048
	$\eta = 4$	623	268	34	0.092
	$\eta = 5$	479	192	25	0.135

対象デバイスのファンクションブロックは, 56 個の Product Term を含む PLA と XOR ゲート D フリップフロップを Macrocell から構成されている [4]. したがって, 表中の Product Term と Macrocell は実装し

た際の回路面積の目安となる値である．表 2, 表 3 より入力 8 bit, 10 bit ともに Dinu らの論理ゲート論理ゲート変換で作成した回路, 提案法で作成した回路は, 乗算器を使用した回路より回路面積を抑えることができている．また提案法が作成した回路は, η の値が増えるにつれて, 誤差は大きくなるものの, Dinu らの論理ゲート変換で作成した回路より回路面積が小さくなっていることがわかる．そして, 8 bit では $\eta = 3$ で乗算器を使用した回路と同程度の誤り率となっており, 回路面積は Product Term が 70% 程度, Macrocell が 80% 程度抑えられている．また 10 bit では $\eta = 4$ で乗算器を使用した回路と同程度の誤り率となっており, 回路面積は Product Term が 36% 程度, Macrocell が 60% 程度抑えられている．

8 bit における乗算器を使用した回路での出力 y が出力を誤ったときの入力 x_1, x_2 をプロットしたグラフを図 6, 提案法 $\eta = 3$ で作成した回路での出力 y が出力を誤ったときの入力 x_1, x_2 をプロットしたグラフを図 7 に示す．図 5, 図 6, 図 7 より提案法で作成した回路においても, 丸め誤差を伴う乗算器を使用した回路と同様に出力境界線付近で出力に誤りが生じている．また, 10 bit の場合についても同様のことが確認できた．

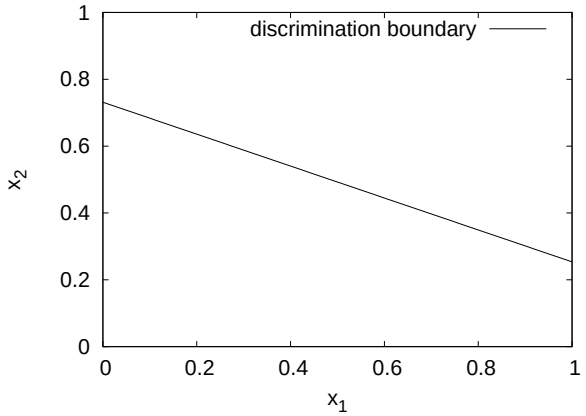

図 5: ニューロンモデルの出力境界線

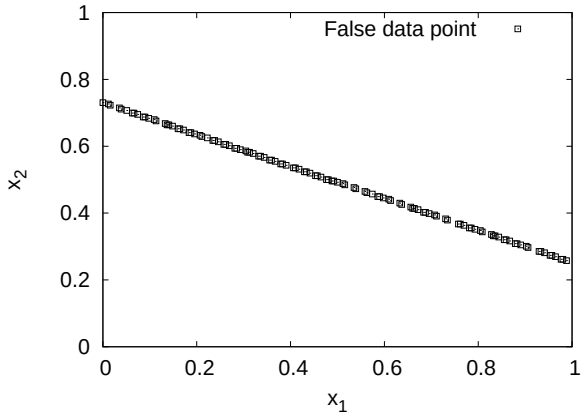

図 6: 入力 8 bit, 乗算器 での出力誤差

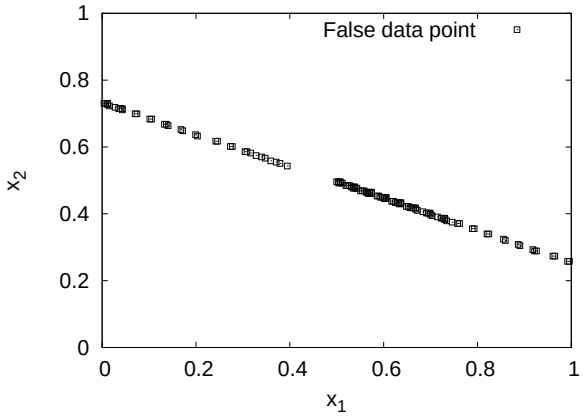

図 7: 入力 8 bit, 提案法 $\eta = 3$ での出力誤差

6. まとめ

面積と精度のトレードオフを考慮したニューラルネットワークのディジタル回路化について検討した．提案法により, 乗算器を用いる手法と同程度の精度で Dinu らの手法より回路面積を抑えたたニューラルネットワークのディジタル回路化を行うことができた．

今後の課題として, bit 数増えた場合でのより効果的な方法の考察, 複雑なニューロンモデルにおける論理ゲート構造への変換, 各種アプリケーションへの実装などがあげられる．

参考文献

[1] 寺村, 宮島, “蓄電池残容量の推定に適したニューラルネットワークとそのシナプス荷重計算法,” 電子情報通信学会論文誌 B, Vol.J92-B, No.4, pp.803-811, 2009.

[2] S. Himavathi, D. Anitha, and A. Muthuramalingam, “Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization,” IEEE Trans. Neural Networks, Vol.18, No.3, pp.880-888, 2007.

[3] A.M. Dinu, M.N. Cirstea, and S.E. Cirstea, “Direct Neural-Network Hardware-Implementation Algorithm,” IEEE Trans. Industrial Electronics, Vol.57, No.5, pp.1845-1848, 2010.

[4] Xilinx Inc., “CoolRunner-Ⅱ CPLD ファミリデータシード”, 2007.